QUANTCONNECT DOCUMENTATION

Learn to use QuantConnect and Explore Features

WRITING ALGORITHMS

Learn tools you need to build algorithmic trading strategies

Build on a mature, flexible, feature-complete API managing billions of dollars capital, and used by 5,000+ investors every month.

Table of Content

- 1 Key Concepts
- 1.1 Getting Started
- 1.2 Algorithm Engine
- 1.3 Multi-Asset Modeling
- 1.4 Time Modeling
- 1.4.1 Periods
- 1.4.2 Timeslices
- 1.4.3 Time Zones
- 1.5 Security Identifiers
- 1.6 Event Handlers
- 1.7 Python and LEAN
- 1.8 Duck Typing
- 1.9 Research Guide
- 1.10 Globals and Statics
- 1.11 Libraries
- 1.12 Debugging Tools
- 1.13 Glossary
- 2 Initialization
- 3 Securities
- 3.1 Key Concepts
- 3.2 Requesting Data
- 3.3 Handling Data
- 3.4 Market Hours
- 3.5 Filtering Data
- 3.6 Asset Classes
- 3.6.1 US Equity
- 3.6.1.1 Requesting Data
- 3.6.1.2 Handling Data
- 3.6.1.3 Corporate Actions
- 3.6.1.4 Corporate Fundamentals
- 3.6.1.5 Shorting
- 3.6.1.6 Data Preparation
- 3.6.1.7 Market Hours
- 3.6.2 Equity Options
- 3.6.2.1 Requesting Data
- 3.6.2.1.1 Universes
- 3.6.2.1.2 Individual Contracts
- 3.6.2.2 Handling Data
- 3.6.2.3 Greeks and Implied Volatility
- 3.6.2.3.1 Key Concepts
- 3.6.2.3.2 Indicators
- 3.6.2.4 Market Hours
- 3.6.3 Crypto
- 3.6.3.1 Requesting Data

- 3.6.3.2 Handling Data
- 3.6.3.3 Holdings
- 3.6.3.4 Market Hours
- 3.6.4 Crypto Futures
- 3.6.4.1 Requesting Data
- 3.6.4.2 Handling Data
- 3.6.4.3 Market Hours
- 3.6.5 Forex
- 3.6.5.1 Requesting Data
- 3.6.5.2 Handling Data
- 3.6.5.3 Market Hours
- 3.6.5.3.1 EURNZD
- 3.6.5.3.2 GBPNZD
- 3.6.5.3.3 NZDCAD
- 3.6.5.3.4 NZDCHF
- 3.6.5.3.5 NZDHKD
- 3.6.5.3.6 NZDJPY
- 3.6.5.3.7 NZDSGD
- 3.6.5.3.8 NZDUSD
- 3.6.6 Futures
- 3.6.6.1 Requesting Data
- 3.6.6.1.1 Universes
- 3.6.6.1.2 Individual Contracts
- 3.6.6.2 Handling Data
- 3.6.6.3 Market Hours
- 3.6.6.3.1 CBOT
- 3.6.6.3.1.1 10Y
- 3.6.6.3.1.2 2YY
- 3.6.6.3.1.3 30Y
- 3.6.6.3.1.4 5YY
- 3.6.6.3.1.5 AW
- 3.6.6.3.1.6 BCF
- 3.6.6.3.1.7 BWF
- 3.6.6.3.1.8 EH
- 3.6.6.3.1.9 F1U
- 3.6.6.3.1.10 KE
- 3.6.6.3.1.11 MYM
- 3.6.6.3.1.12 TN
- 3.6.6.3.1.13 UB
- 3.6.6.3.1.14 YM
- 3.6.6.3.1.15 ZB
- 3.6.6.3.1.16 ZC
- 3.6.6.3.1.17 ZF
- 3.6.6.3.1.18 ZL
- 3.6.6.3.1.19 ZM

- 3.6.6.3.1.20 ZN
- 3.6.6.3.1.21 ZO
- 3.6.6.3.1.22 ZS
- 3.6.6.3.1.23 ZT
- 3.6.6.3.1.24 ZW
- 3.6.6.3.2 CFE
- 3.6.6.3.2.1 VX
- 3.6.6.3.3 CME
- 3.6.6.3.3.1 6A
- 3.6.6.3.3.2 6B
- 3.6.6.3.3.3 6C
- 3.6.6.3.3.4 6E
- 3.6.6.3.3.5 6J
- 3.6.6.3.3.6 6L
- 3.6.6.3.3.7 6M
- 3.6.6.3.3.8 6N
- 3.6.6.3.3.9 6R
- 3.6.6.3.3.10 6S
- 3.6.6.3.3.11 6Z
- 3.6.6.3.3.12 ACD
- 3.6.6.3.3.13 AJY
- 3.6.6.3.3.14 ANE
- 3.6.6.3.3.15 BIO
- 3.6.6.3.3.16 BTC
- 3.6.6.3.3.17 CB
- 3.6.6.3.3.18 CJY
- 3.6.6.3.3.19 CNH
- 3.6.6.3.3.20 CSC
- 3.6.6.3.3.21 DC
- 3.6.6.3.3.22 DY
- 3.6.6.3.3.23 E3G
- 3.6.6.3.3.24 E7
- 3.6.6.3.3.25 EAD
- 3.6.6.3.3.26 ECD
- 3.6.6.3.3.27 EI
- 3.6.6.3.3.28 EMD
- 3.6.6.3.3.29 ENY
- 3.6.6.3.3.30 ES
- 3.6.6.3.3.31 ESG
- 3.6.6.3.3.32 ESK
- 3.6.6.3.3.33 ETH
- 3.6.6.3.3.34 FT1
- 3.6.6.3.3.35 FT5
- 3.6.6.3.3.36 FTU
- 3.6.6.3.3.37 GD

- 3.6.6.3.3.38 GDK
- 3.6.6.3.3.39 GE
- 3.6.6.3.3.40 GF
- 3.6.6.3.3.41 GNF
- 3.6.6.3.3.42 HE
- 3.6.6.3.3.43 IBV
- 3.6.6.3.3.44 J7
- 3.6.6.3.3.45 LBR
- 3.6.6.3.3.46 LBS
- 3.6.6.3.3.47 LE
- 3.6.6.3.3.48 M2K
- 3.6.6.3.3.49 M6A
- 3.6.6.3.3.50 M6B
- 3.6.6.3.3.51 M6C
- 3.6.6.3.3.52 M6E
- 3.6.6.3.3.53 M6J
- 3.6.6.3.3.54 M6S
- 3.6.6.3.3.55 MBT
- 3.6.6.3.3.56 MCD
- 3.6.6.3.3.57 MES
- 3.6.6.3.3.58 MET
- 3.6.6.3.3.59 MIB
- 3.6.6.3.3.60 MIR
- 3.6.6.3.3.61 MJY
- 3.6.6.3.3.62 MNH
- 3.6.6.3.3.63 MNQ
- 3.6.6.3.3.64 MRB
- 3.6.6.3.3.65 MSF
- 3.6.6.3.3.66 NIY
- 3.6.6.3.3.67 NKD
- 3.6.6.3.3.68 NQ
- 3.6.6.3.3.69 RS1
- 3.6.6.3.3.70 RTY
- 3.6.6.3.3.71 RX
- 3.6.6.3.3.72 SDA
- 3.6.6.3.3.73 TPD
- 3.6.6.3.3.74 TPY
- 3.6.6.3.4 COMEX
- 3.6.6.3.4.1 AUP
- 3.6.6.3.4.2 EDP
- 3.6.6.3.4.3 GC
- 3.6.6.3.4.4 HG
- 3.6.6.3.4.5 MGC
- 3.6.6.3.4.6 MGT
- 3.6.6.3.4.7 SI

- 3.6.6.3.4.8 SIL
- 3.6.6.3.5 EUREX
- 3.6.6.3.6 HKFE
- 3.6.6.3.6.1 HSI
- 3.6.6.3.7 ICE
- 3.6.6.3.7.1 B
- 3.6.6.3.7.2 CC
- 3.6.6.3.7.3 CT
- 3.6.6.3.7.4 DX
- 3.6.6.3.7.5 G
- 3.6.6.3.7.6 KC
- 3.6.6.3.7.7 OJ
- 3.6.6.3.7.8 SB
- 3.6.6.3.8 INDIA
- 3.6.6.3.9 NYMEX
- 3.6.6.3.9.1 1S
- 3.6.6.3.9.2 22
- 3.6.6.3.9.3 A0D
- 3.6.6.3.9.4 A0F
- 3.6.6.3.9.5 A1L
- 3.6.6.3.9.6 A1M
- 3.6.6.3.9.7 A1R
- 3.6.6.3.9.8 A32
- 3.6.6.3.9.9 A3G
- 3.6.6.3.9.10 A7E
- 3.6.6.3.9.11 A7I
- 3.6.6.3.9.12 A7Q
- 3.6.6.3.9.13 A8J
- 3.6.6.3.9.14 A8K
- 3.6.6.3.9.15 A8O
- 3.6.6.3.9.16 A91
- 3.6.6.3.9.17 A9N
- 3.6.6.3.9.18 AA6
- 3.6.6.3.9.19 AA8
- 3.6.6.3.9.20 ABS
- 3.6.6.3.9.21 ABT
- 3.6.6.3.9.22 AC0
- 3.6.6.3.9.23 AD0
- 3.6.6.3.9.24 ADB
- 3.6.6.3.9.25 AE5
- 3.6.6.3.9.26 AGA
- 3.6.6.3.9.27 AJL
- 3.6.6.3.9.28 AJS
- 3.6.6.3.9.29 AKL
- 3.6.6.3.9.30 AKZ

- 3.6.6.3.9.31 APS
- 3.6.6.3.9.32 AR0
- 3.6.6.3.9.33 ARE
- 3.6.6.3.9.34 AVZ
- 3.6.6.3.9.35 AYV
- 3.6.6.3.9.36 AYX
- 3.6.6.3.9.37 AZ1
- 3.6.6.3.9.38 B0
- 3.6.6.3.9.39 B7H
- 3.6.6.3.9.40 BK
- 3.6.6.3.9.41 BOO
- 3.6.6.3.9.42 BR7
- 3.6.6.3.9.43 BZ
- 3.6.6.3.9.44 CL
- 3.6.6.3.9.45 CRB
- 3.6.6.3.9.46 CSW
- 3.6.6.3.9.47 CSX
- 3.6.6.3.9.48 CU
- 3.6.6.3.9.49 D1N
- 3.6.6.3.9.50 DCB
- 3.6.6.3.9.51 E6
- 3.6.6.3.9.52 EN
- 3.6.6.3.9.53 EPN
- 3.6.6.3.9.54 EVC
- 3.6.6.3.9.55 EWG
- 3.6.6.3.9.56 EWN
- 3.6.6.3.9.57 EXR
- 3.6.6.3.9.58 FO
- 3.6.6.3.9.59 FRC
- 3.6.6.3.9.60 FSS
- 3.6.6.3.9.61 GCU
- 3.6.6.3.9.62 HCL
- 3.6.6.3.9.63 HH
- 3.6.6.3.9.64 HO
- 3.6.6.3.9.65 HP
- 3.6.6.3.9.66 HRC
- 3.6.6.3.9.67 HTT
- 3.6.6.3.9.68 M1B
- 3.6.6.3.9.69 M35
- 3.6.6.3.9.70 M5F
- 3.6.6.3.9.71 MAF
- 3.6.6.3.9.72 MCL
- 3.6.6.3.9.73 MEF
- 3.6.6.3.9.74 NG
- 3.6.6.3.9.75 PA

- 3.6.6.3.9.76 PAM
- 3.6.6.3.9.77 PL
- 3.6.6.3.9.78 R5O
- 3.6.6.3.9.79 RB
- 3.6.6.3.9.80 S5O
- 3.6.6.3.9.81 YO
- 3.6.6.3.10 NYSELIFFE
- 3.6.6.3.11 SGX
- 3.6.6.3.11.1 IN
- 3.6.6.3.11.2 NK
- 3.6.6.3.11.3 TW
- 3.6.7 Future Options
- 3.6.7.1 Requesting Data
- 3.6.7.1.1 Universes
- 3.6.7.1.2 Individual Contracts
- 3.6.7.2 Handling Data
- 3.6.7.3 Greeks and Implied Volatility
- 3.6.7.3.1 Key Concepts
- 3.6.7.3.2 Indicators
- 3.6.7.4 Market Hours
- 3.6.8 Index
- 3.6.8.1 Requesting Data
- 3.6.8.2 Handling Data
- 3.6.8.3 Market Hours
- 3.6.8.3.1 USA
- 3.6.8.3.1.1 AEX
- 3.6.8.3.1.2 BKX
- 3.6.8.3.1.3 BRR
- 3.6.8.3.1.4 BRTI
- 3.6.8.3.1.5 BXD
- 3.6.8.3.1.6 BXM
- 3.6.8.3.1.7 BXN
- 3.6.8.3.1.8 BXR
- 3.6.8.3.1.9 CEX
- 3.6.8.3.1.10 CLL
- 3.6.8.3.1.11 COMP
- 3.6.8.3.1.12 COR1M
- 3.6.8.3.1.13 COR1Y
- 3.6.8.3.1.14 COR30D
- 3.6.8.3.1.15 COR3M
- 3.6.8.3.1.16 COR6M
- 3.6.8.3.1.17 COR9M
- 3.6.8.3.1.18 DJCIAGC
- 3.6.8.3.1.19 DJCICC
- 3.6.8.3.1.20 DJCIGC

- 3.6.8.3.1.21 DJCIGR
- 3.6.8.3.1.22 DJCIIK
- 3.6.8.3.1.23 DJCIKC
- 3.6.8.3.1.24 DJCISB
- 3.6.8.3.1.25 DJCISI
- 3.6.8.3.1.26 DJI
- 3.6.8.3.1.27 DJINET
- 3.6.8.3.1.28 DJR
- 3.6.8.3.1.29 DJX
- 3.6.8.3.1.30 DRG
- 3.6.8.3.1.31 DTX
- 3.6.8.3.1.32 DUX
- 3.6.8.3.1.33 DVS
- 3.6.8.3.1.34 DWCF
- 3.6.8.3.1.35 DWCPF
- 3.6.8.3.1.36 DXL
- 3.6.8.3.1.37 DXY
- 3.6.8.3.1.38 EVZ
- 3.6.8.3.1.39 FTW5000
- 3.6.8.3.1.40 FVX
- 3.6.8.3.1.41 GVZ
- 3.6.8.3.1.42 HGX
- 3.6.8.3.1.43 MID
- 3.6.8.3.1.44 MIDG
- 3.6.8.3.1.45 MIDV
- 3.6.8.3.1.46 MRUT
- 3.6.8.3.1.47 NDX
- 3.6.8.3.1.48 NYA
- 3.6.8.3.1.49 NYFANG
- 3.6.8.3.1.50 NYXBT
- 3.6.8.3.1.51 OEX
- 3.6.8.3.1.52 OSX
- 3.6.8.3.1.53 OVX
- 3.6.8.3.1.54 PUT
- 3.6.8.3.1.55 RLS
- 3.6.8.3.1.56 RUA
- 3.6.8.3.1.57 RUI
- 3.6.8.3.1.58 RUT
- 3.6.8.3.1.59 RVX
- 3.6.8.3.1.60 SET
- 3.6.8.3.1.61 SGX
- 3.6.8.3.1.62 SKEW
- 3.6.8.3.1.63 SMLG
- 3.6.8.3.1.64 SP600
- 3.6.8.3.1.65 SPGSCI

- 3.6.8.3.1.66 SPSIBI
- 3.6.8.3.1.67 SPSV
- 3.6.8.3.1.68 SPX
- 3.6.8.3.1.69 SVX
- 3.6.8.3.1.70 TNX
- 3.6.8.3.1.71 TYX
- 3.6.8.3.1.72 UKX
- 3.6.8.3.1.73 UTIL
- 3.6.8.3.1.74 UTY
- 3.6.8.3.1.75 VAF
- 3.6.8.3.1.76 VIF
- 3.6.8.3.1.77 VIN
- 3.6.8.3.1.78 VIX
- 3.6.8.3.1.79 VIX1D
- 3.6.8.3.1.80 VIX1Y
- 3.6.8.3.1.81 VIX3M
- 3.6.8.3.1.82 VIX6M
- 3.6.8.3.1.83 VIX9D
- 3.6.8.3.1.84 VOLI
- 3.6.8.3.1.85 VPD
- 3.6.8.3.1.86 VPN
- 3.6.8.3.1.87 VRO
- 3.6.8.3.1.88 VVIX
- 3.6.8.3.1.89 VWA
- 3.6.8.3.1.90 VWB
- 3.6.8.3.1.91 VXD
- 3.6.8.3.1.92 VXN
- 3.6.8.3.1.93 VXO
- 3.6.8.3.1.94 VXSLV
- 3.6.8.3.1.95 VXTH
- 3.6.8.3.1.96 VXTLT
- 3.6.8.3.1.97 XAU
- 3.6.8.3.1.98 XDA
- 3.6.8.3.1.99 XDB
- 3.6.8.3.1.100 XEO
- 3.6.8.3.1.101 XMI
- 3.6.8.3.1.102 XNDX
- 3.6.8.3.1.103 XSP
- 3.6.8.3.2 EUREX
- 3.6.8.3.2.1 DAX
- 3.6.8.3.3 HKFE
- 3.6.8.3.3.1 HSI
- 3.6.8.3.4 OSE
- 3.6.8.3.4.1 N225
- 3.6.9 Index Options

- 3.6.9.1 Requesting Data
- 3.6.9.1.1 Universes
- 3.6.9.1.2 Individual Contracts
- 3.6.9.2 Handling Data
- 3.6.9.3 Greeks and Implied Volatility
- 3.6.9.3.1 Key Concepts
- 3.6.9.3.2 Indicators
- 3.6.9.4 Market Hours
- 3.6.9.4.1 USA
- 3.6.9.4.1.1 NDX
- 3.6.9.4.1.2 RUT
- 3.6.9.4.1.3 SPX
- 3.6.9.4.1.4 VIX
- 3.6.10 CFD
- 3.6.10.1 Requesting Data
- 3.6.10.2 Handling Data
- 3.6.10.3 Market Hours
- 3.6.10.3.1 Interactive Brokers
- 3.6.10.3.1.1 AUDCAD
- 3.6.10.3.1.2 AUDCHF
- 3.6.10.3.1.3 AUDCNH
- 3.6.10.3.1.4 AUDHKD
- 3.6.10.3.1.5 AUDJPY
- 3.6.10.3.1.6 AUDNZD
- 3.6.10.3.1.7 AUDSGD
- 3.6.10.3.1.8 AUDUSD
- 3.6.10.3.1.9 AUDZAR
- 3.6.10.3.1.10 CADCHF
- 3.6.10.3.1.11 CADCNH
- 3.6.10.3.1.12 CADHKD
- 3.6.10.3.1.13 CADJPY
- 3.6.10.3.1.14 CHFCNH
- 3.6.10.3.1.15 CHFCZK
- 3.6.10.3.1.16 CHFDKK
- 3.6.10.3.1.17 CHFHUF
- 3.6.10.3.1.18 CHFJPY
- 3.6.10.3.1.19 CHFNOK
- 3.6.10.3.1.20 CHFPLN
- 3.6.10.3.1.21 CHFSEK
- 3.6.10.3.1.22 CHFZAR
- 3.6.10.3.1.23 CNHHKD
- 3.6.10.3.1.24 CNHJPY
- 3.6.10.3.1.25 DKKJPY
- 3.6.10.3.1.26 DKKNOK
- 3.6.10.3.1.27 DKKSEK

- 3.6.10.3.1.28 EURAUD
- 3.6.10.3.1.29 EURCAD
- 3.6.10.3.1.30 EURCHF
- 3.6.10.3.1.31 EURCNH
- 3.6.10.3.1.32 EURCZK
- 3.6.10.3.1.33 EURDKK
- 3.6.10.3.1.34 EURGBP
- 3.6.10.3.1.35 EURHKD
- 3.6.10.3.1.36 EURHUF
- 3.6.10.3.1.37 EURILS
- 3.6.10.3.1.38 EURJPY
- 3.6.10.3.1.39 EURMXN
- 3.6.10.3.1.40 EURNOK
- 3.6.10.3.1.41 EURNZD
- 3.6.10.3.1.42 EURPLN
- 3.6.10.3.1.43 EURRUB
- 3.6.10.3.1.44 EURSEK
- 3.6.10.3.1.45 EURSGD
- 3.6.10.3.1.46 EURUSD
- 3.6.10.3.1.47 EURZAR
- 3.6.10.3.1.48 GBPAUD
- 3.6.10.3.1.49 GBPCAD
- 3.6.10.3.1.50 GBPCHF
- 3.6.10.3.1.51 GBPCNH
- 3.6.10.3.1.52 GBPCZK
- 3.6.10.3.1.53 GBPDKK
- 3.6.10.3.1.54 GBPHKD
- 3.6.10.3.1.55 GBPHUF
- 3.6.10.3.1.56 GBPJPY
- 3.6.10.3.1.57 GBPMXN
- 3.6.10.3.1.58 GBPNOK
- 3.6.10.3.1.59 GBPNZD
- 3.6.10.3.1.60 GBPPLN
- 3.6.10.3.1.61 GBPSEK
- 3.6.10.3.1.62 GBPSGD
- 3.6.10.3.1.63 GBPUSD
- 3.6.10.3.1.64 GBPZAR
- 3.6.10.3.1.65 HKDJPY
- 3.6.10.3.1.66 IBAU200
- 3.6.10.3.1.67 IBCH20
- 3.6.10.3.1.68 IBDE40
- 3.6.10.3.1.69 IBES35
- 3.6.10.3.1.70 IBEU50
- 3.6.10.3.1.71 IBFR40
- 3.6.10.3.1.72 IBGB100

- 3.6.10.3.1.73 IBHK50
- 3.6.10.3.1.74 IBJP225
- 3.6.10.3.1.75 IBNL25
- 3.6.10.3.1.76 IBUS30
- 3.6.10.3.1.77 IBUS500
- 3.6.10.3.1.78 IBUST100
- 3.6.10.3.1.79 MXNJPY
- 3.6.10.3.1.80 NOKJPY
- 3.6.10.3.1.81 NOKSEK
- 3.6.10.3.1.82 NZDCAD
- 3.6.10.3.1.83 NZDCHF
- 3.6.10.3.1.84 NZDJPY
- 3.6.10.3.1.85 NZDUSD
- 3.6.10.3.1.86 SEKJPY
- 3.6.10.3.1.87 SGDCNH
- 3.6.10.3.1.88 SGDJPY
- 3.6.10.3.1.89 USDCAD
- 3.6.10.3.1.90 USDCHF
- 3.6.10.3.1.91 USDCNH
- 3.6.10.3.1.92 USDCZK
- 3.6.10.3.1.93 USDDKK
- 3.6.10.3.1.94 USDHKD
- 3.6.10.3.1.95 USDHUF
- 3.6.10.3.1.96 USDILS
- 3.6.10.3.1.97 USDJPY
- 3.6.10.3.1.98 USDMXN
- 3.6.10.3.1.99 USDNOK
- 3.6.10.3.1.100 USDPLN
- 3.6.10.3.1.101 USDRUB
- 3.6.10.3.1.102 USDSEK
- 3.6.10.3.1.103 USDSGD
- 3.6.10.3.1.104 USDZAR
- 3.6.10.3.1.105 XAGUSD
- 3.6.10.3.1.106 XAUUSD
- 3.6.10.3.1.107 ZARJPY
- 3.6.10.3.2 Oanda
- 3.6.10.3.2.1 AU200AUD
- 3.6.10.3.2.2 BCOUSD
- 3.6.10.3.2.3 CH20CHF
- 3.6.10.3.2.4 CORNUSD
- 3.6.10.3.2.5 DE10YBEUR
- 3.6.10.3.2.6 DE30EUR
- 3.6.10.3.2.7 EU50EUR
- 3.6.10.3.2.8 FR40EUR
- 3.6.10.3.2.9 HK33HKD

- 3.6.10.3.2.10 JP225USD
- 3.6.10.3.2.11 NAS100USD
- 3.6.10.3.2.12 NATGASUSD
- 3.6.10.3.2.13 NL25EUR
- 3.6.10.3.2.14 SG30SGD
- 3.6.10.3.2.15 SOYBNUSD
- 3.6.10.3.2.16 SPX500USD
- 3.6.10.3.2.17 SUGARUSD
- 3.6.10.3.2.18 UK100GBP
- 3.6.10.3.2.19 UK10YBGBP
- 3.6.10.3.2.20 US2000USD
- 3.6.10.3.2.21 US30USD
- 3.6.10.3.2.22 USB02YUSD
- 3.6.10.3.2.23 USB05YUSD
- 3.6.10.3.2.24 USB10YUSD
- 3.6.10.3.2.25 USB30YUSD
- 3.6.10.3.2.26 WHEATUSD
- 3.6.10.3.2.27 WTICOUSD
- 3.6.10.3.2.28 XAGAUD
- 3.6.10.3.2.29 XAGCAD
- 3.6.10.3.2.30 XAGCHF
- 3.6.10.3.2.31 XAGEUR
- 3.6.10.3.2.32 XAGGBP
- 3.6.10.3.2.33 XAGHKD
- 3.6.10.3.2.34 XAGJPY
- 3.6.10.3.2.35 XAGNZD
- 3.6.10.3.2.36 XAGSGD
- 3.6.10.3.2.37 XAGUSD
- 3.6.10.3.2.38 XAUAUD
- 3.6.10.3.2.39 XAUCAD
- 3.6.10.3.2.40 XAUCHF
- 3.6.10.3.2.41 XAUEUR
- 3.6.10.3.2.42 XAUGBP
- 3.6.10.3.2.43 XAUHKD
- 3.6.10.3.2.44 XAUJPY
- 3.6.10.3.2.45 XAUNZD
- 3.6.10.3.2.46 XAUSGD
- 3.6.10.3.2.47 XAUUSD
- 3.6.10.3.2.48 XAUXAG
- 3.6.10.3.2.49 XCUUSD
- 3.6.10.3.2.50 XPDUSD
- 3.6.10.3.2.51 XPTUSD
- 3.6.11 India Equity
- 3.6.11.1 Market Hours
- 4 Portfolio

- 4.1 Key Concepts
- 4.2 Holdings
- 4.3 Cashbook
- 5 Universes
- 5.1 Key Concepts
- 5.2 Settings
- 5.3 Equity
- 5.3.1 Liquidity Universes
- 5.3.2 Fundamental Universes
- 5.3.3 ETF Constituents Universes
- 5.3.4 Chained Universes
- 5.3.5 Legacy Fundamental Universes
- 5.3.6 Alternative Data Universes
- 5.4 Equity Options
- 5.5 Equity CFD
- 5.6 Futures
- 5.7 Future Options
- 5.8 Index Options
- 5.9 Crypto
- 5.10 Custom Universes
- 5.11 Dataless Scheduled Universes
- 6 Datasets
- 6.1 Overview
- 6.2 QuantConnect
- 6.2.1 Binance Crypto Future Margin Rate Data
- 6.2.2 Bybit Crypto Future Margin Rate Data
- 6.2.3 CFD Data
- 6.2.4 Cash Indices
- 6.2.5 FOREX Data
- 6.2.6 Fear and Greed
- 6.2.7 International Future Universe
- 6.2.8 US ETF Constituents
- 6.2.9 US Equities Short Availability
- 6.2.10 US Equity Coarse Universe
- 6.2.11 US Equity Option Universe
- 6.2.12 US Equity Security Master
- 6.2.13 US Future Option Universe
- 6.2.14 US Future Universe
- 6.2.15 US Futures Security Master
- 6.2.16 US Index Option Universe
- 6.3 AlgoSeek
- 6.3.1 US Equities
- 6.3.2 US Equity Options
- 6.3.3 US Future Options
- 6.3.4 US Futures

- 6.3.5 US Index Options
- 6.4 Morningstar
- 6.4.1 US Fundamental Data
- 6.5 TickData
- 6.5.1 International Futures
- 6.6 CoinAPI
- 6.6.1 Binance Crypto Future Price Data
- 6.6.2 Binance Crypto Price Data
- 6.6.3 Binance US Crypto Price Data
- 6.6.4 Bitfinex Crypto Price Data
- 6.6.5 Bybit Crypto Future Price Data
- 6.6.6 Bybit Crypto Price Data
- 6.6.7 Coinbase Crypto Price Data
- 6.6.8 Kraken Crypto Price Data
- 6.7 Benzinga
- 6.7.1 Benzinga News Feed
- 6.8 Blockchain
- 6.8.1 Bitcoin Metadata
- 6.9 Brain
- 6.9.1 Brain Language Metrics on Company Filings
- 6.9.2 Brain ML Stock Ranking
- 6.9.3 Brain Sentiment Indicator
- 6.9.4 Brain Wikipedia Page Views
- 6.10 CoinGecko
- 6.10.1 Crypto Market Cap
- 6.11 EOD Historical Data
- 6.11.1 Economic Events
- 6.11.2 Macroeconomics Indicators
- 6.11.3 Upcoming Dividends
- 6.11.4 Upcoming Earnings
- 6.11.5 Upcoming IPOs
- 6.11.6 Upcoming Splits
- 6.12 Energy Information Administration
- 6.12.1 US Energy Information Administration (EIA)
- 6.13 ExtractAlpha
- 6.13.1 Cross Asset Model
- 6.13.2 Estimize
- 6.13.3 Tactical
- 6.13.4 True Beats
- 6.14 FRED
- 6.14.1 US Federal Reserve (FRED)
- 6.15 Federal Reserve Bank of St Louis
- 6.15.1 US Interest Rate
- 6.16 Kavout
- 6.16.1 Composite Factor Bundle

- 6.17 Nasdaq
- 6.17.1 Data Link
- 6.18 Quiver Quantitative
- 6.18.1 CNBC Trading
- 6.18.2 Corporate Lobbying
- 6.18.3 Insider Trading
- 6.18.4 US Congress Trading
- 6.18.5 US Government Contracts
- 6.18.6 WallStreetBets
- 6.19 RegAlytics
- 6.19.1 US Regulatory Alerts - Financial Sector
- 6.20 Securities and Exchange Commission
- 6.20.1 US SEC Filings
- 6.21 Smart Insider
- 6.21.1 Corporate Buybacks
- 6.22 Tiingo
- 6.22.1 Tiingo News Feed
- 6.23 Treasury Department
- 6.23.1 US Treasury Yield Curve
- 6.24 VIX Central
- 6.24.1 VIX Central Contango
- 7 Importing Data
- 7.1 Key Concepts
- 7.2 Streaming Data
- 7.2.1 Key Concepts
- 7.2.2 Custom Securities
- 7.2.2.1 Key Concepts
- 7.2.2.2 CSV Format Example
- 7.2.2.3 JSON Format Example
- 7.2.3 Custom Universes
- 7.2.3.1 Key Concepts
- 7.2.3.2 CSV Format Example
- 7.2.3.3 JSON Format Example
- 7.3 Bulk Downloads
- 8 Consolidating Data
- 8.1 Getting Started
- 8.2 Consolidator Types
- 8.2.1 Time Period Consolidators
- 8.2.2 Calendar Consolidators
- 8.2.3 Count Consolidators
- 8.2.4 Mixed-Mode Consolidators
- 8.2.5 Renko Consolidators
- 8.2.5.1 Renko Consolidators
- 8.2.5.2 Classic Renko Consolidators
- 8.2.5.3 Volume Renko Consolidators

- 8.2.6 Range Consolidators
- 8.2.6.1 Range Consolidators
- 8.2.6.2 Classic Range Consolidators
- 8.2.7 Combining Consolidators
- 8.3 Consolidator History
- 8.4 Updating Indicators
- 9 Historical Data
- 9.1 Getting Started
- 9.2 History Requests
- 9.3 History Responses
- 9.4 Asset Classes
- 9.4.1 US Equities
- 9.4.2 Equity Options
- 9.4.3 Crypto
- 9.4.4 Crypto Futures
- 9.4.5 Forex
- 9.4.6 Futures
- 9.4.7 Future Options
- 9.4.8 Index
- 9.4.9 Index Options
- 9.4.10 CFD
- 9.5 Alternative Data
- 9.6 Custom Data
- 9.7 Universe Data
- 9.8 Warm Up Periods
- 9.9 Rolling Window
- 9.10 Common Errors
- 9.11 Live Trading
- 10 Trading and Orders
- 10.1 Key Concepts
- 10.2 Order Management
- 10.2.1 Order Tickets
- 10.2.2 Transaction Manager
- 10.3 Order Types
- 10.3.1 Market Orders
- 10.3.2 Limit Orders
- 10.3.3 Limit if Touched Orders
- 10.3.4 Stop Market Orders
- 10.3.5 Stop Limit Orders
- 10.3.6 Trailing Stop Orders
- 10.3.7 Market On Open Orders
- 10.3.8 Market On Close Orders
- 10.3.9 Combo Market Orders
- 10.3.10 Combo Limit Orders
- 10.3.11 Combo Leg Limit Orders

- 10.3.12 Option Exercise Orders
- 10.3.13 Other Order Types
- 10.4 Position Sizing
- 10.5 Liquidating Positions
- 10.6 Crypto Trades
- 10.7 Option Strategies
- 10.7.1 Bear Call Spread
- 10.7.2 Bear Put Spread
- 10.7.3 Bull Call Spread
- 10.7.4 Bull Put Spread
- 10.7.5 Long Call Butterfly
- 10.7.6 Short Call Butterfly
- 10.7.7 Long Put Butterfly
- 10.7.8 Short Put Butterfly
- 10.7.9 Long Call Calendar Spread
- 10.7.10 Short Call Calendar Spread
- 10.7.11 Long Put Calendar Spread
- 10.7.12 Short Put Calendar Spread
- 10.7.13 Covered Call
- 10.7.14 Covered Put
- 10.7.15 Long Iron Butterfly
- 10.7.16 Short Iron Butterfly
- 10.7.17 Long Iron Condor
- 10.7.18 Short Iron Condor
- 10.7.19 Naked Call
- 10.7.20 Naked Put
- 10.7.21 Protective Call
- 10.7.22 Protective Put
- 10.7.23 Protective Collar
- 10.7.24 Long Straddle
- 10.7.25 Short Straddle
- 10.7.26 Long Strangle
- 10.7.27 Short Strangle
- 10.7.28 Conversion
- 10.7.29 Reverse Conversion
- 10.7.30 Long Box Spread
- 10.7.31 Short Box Spread
- 10.7.32 Long Jelly Roll
- 10.7.33 Short Jelly Roll
- 10.7.34 Bear Call Ladder
- 10.7.35 Bear Put Ladder
- 10.7.36 Bull Call Ladder
- 10.7.37 Bull Put Ladder
- 10.7.38 Long Call Backspread
- 10.7.39 Short Call Backspread

- 10.7.40 Long Put Backspread
- 10.7.41 Short Put Backspread
- 10.8 Order Properties
- 10.9 Order Events
- 10.10 Order Errors
- 10.11 Trade Statistics
- 10.12 Trading Calendar
- 10.13 Financial Advisors
- 11 Reality Modeling
- 11.1 Key Concepts
- 11.2 Trade Fills
- 11.2.1 Key Concepts
- 11.2.2 Supported Models
- 11.2.2.1 Equity Model
- 11.2.2.2 Future Model
- 11.2.2.3 Future Option Model
- 11.2.2.4 Immediate Model
- 11.2.2.5 Latest Price Model
- 11.3 Slippage
- 11.3.1 Key Concepts
- 11.3.2 Supported Models
- 11.4 Transaction Fees
- 11.4.1 Key Concepts
- 11.4.2 Supported Models
- 11.5 Brokerages
- 11.5.1 Key Concepts
- 11.5.2 Supported Models
- 11.5.2.1 QuantConnect Paper Trading
- 11.5.2.2 Interactive Brokers
- 11.5.2.3 TradeStation
- 11.5.2.4 Tastytrade
- 11.5.2.5 Alpaca
- 11.5.2.6 Charles Schwab
- 11.5.2.7 Binance
- 11.5.2.8 Bybit
- 11.5.2.9 Tradier
- 11.5.2.10 Kraken
- 11.5.2.11 Coinbase
- 11.5.2.12 Bitfinex
- 11.5.2.13 dYdX
- 11.5.2.14 Terminal Link
- 11.5.2.15 SSC Eze
- 11.5.2.16 Trading Technologies
- 11.5.2.17 Wolverine
- 11.5.2.18 Oanda

- 11.6 Brokerage Message Handler
- 11.7 Buying Power
- 11.8 Settlement
- 11.8.1 Key Concepts
- 11.8.2 Supported Models
- 11.9 Options Models
- 11.9.1 Pricing
- 11.9.2 Volatility
- 11.9.2.1 Key Concepts
- 11.9.2.2 Supported Models
- 11.9.3 Exercise
- 11.9.4 Assignment
- 11.10 Risk Free Interest Rate
- 11.10.1 Key Concepts
- 11.10.2 Supported Models
- 11.11 Dividend Yield
- 11.11.1 Key Concepts
- 11.11.2 Supported Models
- 11.12 Margin Interest Rate
- 11.12.1 Key Concepts
- 11.12.2 Supported Models
- 11.13 Margin Calls
- 11.14 Short Availability
- 11.14.1 Key Concepts
- 11.14.2 Supported Providers
- 12 Scheduled Events
- 13 Indicators
- 13.1 Supported Indicators
- 13.2 Key Concepts
- 13.3 Manual Indicators
- 13.4 Automatic Indicators
- 13.5 Plotting Indicators
- 13.6 Combining Indicators
- 13.7 Custom Indicators
- 13.8 Indicator Universes
- 13.9 Rolling Window
- 14 Object Store
- 15 Optimization
- 15.1 Parameters
- 15.2 Walk Forward Optimization
- 16 Machine Learning
- 16.1 Key Concepts
- 16.2 Training Models
- 16.3 Popular Libraries
- 16.3.1 Aesera

- 16.3.2 Copula
- 16.3.3 GPlearn
- 16.3.4 Hmmlearn
- 16.3.5 Keras
- 16.3.6 PyTorch
- 16.3.7 Scikit-Learn
- 16.3.8 Tensorflow
- 16.3.9 Tslearn
- 16.3.10 XGBoost
- 16.4 Hugging Face
- 16.4.1 Key Concepts
- 16.4.2 Popular Models
- 16.4.2.1 Chronos-T5
- 16.4.2.2 DistilBERT
- 16.4.2.3 FinBERT
- 17 Algorithm Framework
- 17.1 Overview
- 17.2 Universe Selection
- 17.2.1 Key Concepts
- 17.2.2 Universe Settings
- 17.2.3 Manual Universes
- 17.2.4 Fundamental Universes
- 17.2.5 ETF Constituents Universes
- 17.2.6 Scheduled Universes
- 17.2.7 Futures Universes
- 17.2.8 Options Universes
- 17.2.9 Legacy Fundamental Universes
- 17.3 Alpha
- 17.3.1 Key Concepts
- 17.3.2 Supported Models
- 17.4 Portfolio Construction
- 17.4.1 Key Concepts
- 17.4.2 Supported Models
- 17.4.3 Supported Optimizers
- 17.5 Risk Management
- 17.5.1 Key Concepts
- 17.5.2 Supported Models
- 17.6 Execution
- 17.6.1 Key Concepts
- 17.6.2 Supported Models
- 17.7 Hybrid Algorithms
- 17.8 Insight Manager
- 18 Charting
- 19 Logging
- 20 Statistics

- 20.1 Algorithm Statistics
- 20.2 Runtime Statistics
- 20.3 Trade Statistics
- 21 Live Trading
- 21.1 Key Concepts
- 21.2 Brokerages
- 21.3 Data Providers
- 21.4 Trading and Orders
- 21.4.1 Financial Advisors
- 21.5 Reconciliation
- 21.6 Notifications
- 21.7 Commands
- 21.8 Arbitrage
- 21.9 Charting and Logging
- 21.10 Signal Exports
- 21.10.1 Key Concepts
- 21.10.2 Collective2
- 21.10.3 Numerai
- 21.10.4 vBase
- 22 Strategy Library
- 23 API Reference
- 24 Migrations
- 24.1 Zipline
- 24.1.1 Initialization
- 24.1.2 Using Data
- 24.1.3 Ordering
- 24.1.4 Logging and Plotting
- 24.1.5 Quick Reference

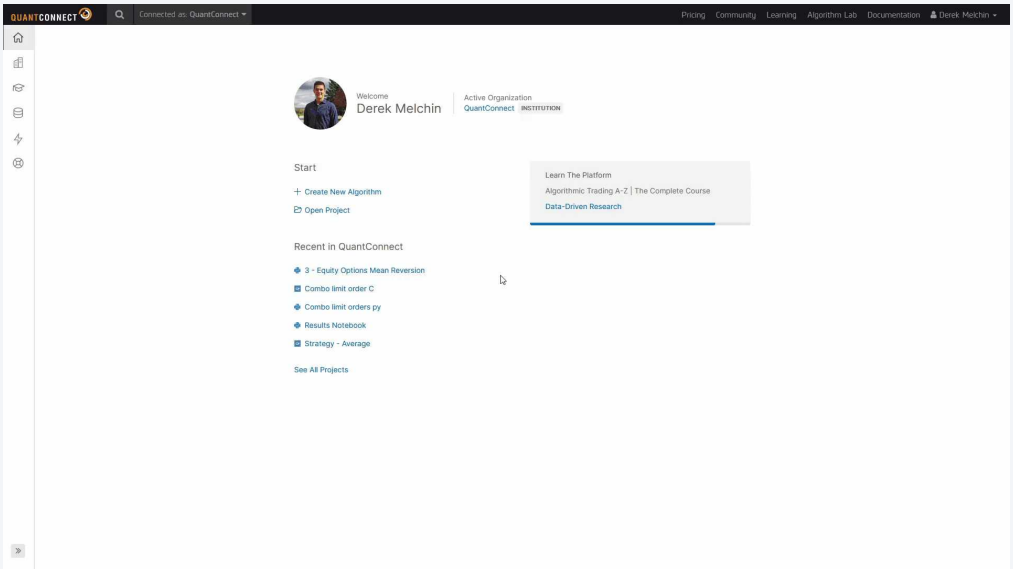
Key Concepts

Key Concepts

Getting Started

GET STARTED WITH WRITING ALGORITHMS

Guide through creating a project, running your first backtest, and live algo trading.



Follow these steps to create, backtest, and paper trade a new algorithm:

1. Create a new project.

The process to creating a new project depends on if you use the Cloud Platform , Local Platform , or CLI .

2. Backtest the default algorithm.

The process to backtest the algorithm depends on if you use the Cloud Platform , Local Platform , or CLI .

3. Deploy the algorithm to live paper trading.

The process to deploy the algorithm live depends on if you use the Cloud Platform , Local Platform , or CLI .

Key Concepts

Algorithm Engine

Introduction

LEAN Engine is an open-source algorithmic trading engine built for easy strategy research, backtesting, and live trading. We integrate with common data providers and brokerages, so you can quickly deploy algorithmic trading strategies. The core of the LEAN Engine is written in C#, but it operates seamlessly on Linux, Mac and Windows operating systems. To use it, you can write algorithms in Python 3.11 or C#. QuantConnect maintains the LEAN project and uses it to drive the web-based algorithmic trading platform on the website.

Since LEAN is open-source, you aren't locked-in to the QuantConnect platform. You can run LEAN with your own infrastructure, data, and brokerage connections. If you use QuantConnect, our team of engineers manage the infrastructure, you can run the latest version of LEAN with all of its brokerage connections, and you can utilize the datasets in the Dataset Market .

Your Algorithm and LEAN

To create a trading algorithm with LEAN, define a subclass of the `QCAlgorithm` class. LEAN loads your algorithm into the engine during the building and compilation process. LEAN runs an algorithm manager that synchronizes the data your algorithm requests, injects the data into your algorithm so you can place trades, processes your orders, and then updates your algorithm state.

The LEAN engine manages your portfolio and data feeds, so you focus on your algorithm strategy and execution. We automatically provide basic portfolio management and reality modeling underneath the hood. The `QCAlgorithm` class provides some key helper properties for you to use, including the Security Manager, Portfolio Manager, Transactions Manager, Notification Manager, and Scheduling Manager. The class also has hundreds of helper methods to make the API easy to use.

The Securities property is a dictionary of `Security` objects. Each asset (Equity, Forex pair, etc) in your algorithm has a `Security` object. All the models for a security live on these objects. For example, `Securities["IBM"].FeeModel` and `Securities["IBM"].Price` return the fee model and price of IBM, respectively.

The Portfolio is a dictionary of `SecurityHolding` objects. These classes track the profits, losses, fees, and quantity of individual portfolio holdings. For example, `Portfolio["IBM"].LastTradeProfit` returns the profit of your last IBM trade.

Other helpers like Transactions , Schedule , Notify , and Universe have their own helper methods.

```C#
public class QCAlgorithm
{
    SecurityManager Securities;               // Array of Security objects.
    SecurityPortfolioManager Portfolio;       // Array of SecurityHolding objects
    SecurityTransactionManager Transactions;  // Transactions helper
    ScheduleManager Schedule;                 // Scheduling helper
    NotificationManager Notify;               // Email, SMS helper
    UniverseManager Universe;                 // Universe helper

    // Set up Requested Data, Cash, Time Period.
    public virtual void Initialize() { ... };

    // Event Handlers:
    public virtual void OnData(Slice slice) { ... };
    public virtual void OnEndOfDay(Symbol symbol) { ... };
    public virtual void OnEndOfAlgorithm() { ... };

    // Indicator Helpers
    public SimpleMovingAverage SMA(Symbol symbol, int period) { ... };
}
```

Only define one algorithm class per project. To trade multiple strategies in a single algorithm, implement an Algorithm Framework design with multiple Alpha models . To be notified when you can define more than one algorithm class per project, subscribe to GitHub Issue #6968 .

Threads in LEAN

LEAN is multi-threaded and attempts to consume as much CPU as possible to perform given work as quickly as possible. The analysis engine loads data parallel and synchronizes it to generate the requested security information.

The client algorithm is plugged into LEAN, and has its events triggered synchronously in backtesting. The primary bottle neck to LEAN execution is executing client code.

In live trading, most events are synchronous as in backtesting, however order events are triggered immediately from the brokerage thread (asynchronously) to ensure the lowest latency. For example; a strategy using hourly data, with a limit order could fill between data bars. This fill event is triggered when it occurs in live trading.

We recommend using thread safe collections when possible, or locks around collections to ensure they are thread-safe.

Batch vs Stream Analysis

Backtesting platforms come in two general varieties, batch processing and event streaming. Batch processing backtesting is much simpler. It loads all data into an array and passes it to your algorithm for analysis. Because your algorithm has access to future data points, it is easy to introduce look-ahead bias . Most home-grown analysis tools are batch systems.

QuantConnect/LEAN is a streaming analysis system. In live trading, your algorithm receives data points one after another over time. QuantConnect models this in backtesting, streaming data to your algorithm in fast-forward mode. Because of this, you can't access price data beyond the Time Frontier . Although streaming analysis is slightly trickier to understand, it allows your algorithm to seamlessly work in backtests and live trading with no code changes.

Event Flow

When you deploy an algorithm, LEAN first calls the Initialize method. In live mode, the engine loads your holdings and open orders from your brokerage account to add data subscriptions and populate the `Securities` , `Portfolio` , and `Transactions` objects. LEAN receives the data from the subscriptions, synchronizes the data to create a timeslice , and then performs the following steps. Your algorithm can spend up to 10 minutes on each timeslice unless you call the Train method.

1. If it's a backtest, check if there are Scheduled Events in the past that didn't fire because there was no data between the

previous slice and the current slice. LEAN automatically creates a Scheduled Event to call the `OnEndOfDay` method at the end of each day.

In live mode, Scheduled Events occur in a separate thread from the algorithm manager, so they run at the correct time.

2. Update the algorithm time.
3. Update the `CurrentSlice` .
4. Pass the `SymbolChangedEvents` to the `OnSymbolChangedEvents` method.
5. Cancel all open orders for securities that changed their ticker .
6. Add a `Security` object to the `Securities` collection for each new security in the universe.
7. Update the `Security` objects with the latest data.
8. Update the `Cash` objects in the CashBook with the latest data.
9. Process fill models for non-market orders.
10. Submit market on open orders to liquidate Equity Option contracts if the underlying Equity has a split warning .
11. Process margin calls .
12. If it's time to settle unsettled cash , perform settlement.
13. Call the OnSecuritiesChanged method with the latest security changes.
14. Apply dividends to the portfolio.
15. For securities that have a split warning, update their portfolio holdings and adjust their open orders to account for the split.
16. Update consolidators with the latest data.
17. Pass custom data to the `OnData` method.
18. Pass the `Dividends` to the `OnDividends` method.
19. Pass the `Splits` to the `OnSplits` method.
20. Pass the `Delistings` to the `OnDelistings` method.
21. Pass the `Slice` to the `OnData` method.
22. Perform universe selection .
23. Pass the `Slice` to the `Update` method of each Alpha model .
24. Pass the `Insight` objects from the Alpha model to the Portfolio Construction model .
25. Pass the `PortfolioTarget` objects from the Portfolio Construction model to each Risk Management model .
26. Pass the risk-adjusted `PortfolioTarget` objects from the Risk Management models to the Execution model .

When your algorithm stops executing, LEAN calls the `OnEndOfAlgorithm` method .

Python Support

The LEAN engine is written in C#, but you can create algorithms in C# or Python. If you program in Python, LEAN uses Python.Net to bridge between the C# engine and your algorithm. It can be slow to move from Python to C#. If you access C# objects in your Python algorithm, it's fastest to only access them once and save a reference if you need to access them again.

Key Concepts

Multi-Asset Modeling

Introduction

We designed LEAN as a multi-asset platform with out-of-the-box support for multiple securities, so you can model complex portfolios like hedge funds.

Asset Portfolio

The portfolio manages the individual securities it contains. It tracks the cost of holding each security. It aggregates the performance of the individual securities in the portfolio to produce statistics like net profit and drawdown . The portfolio also holds information about each currency in its cashbook.

Cashbooks

We designed LEAN to be a multi-currency platform. LEAN can trade Forex, Cryptocurrencies, and other assets that are quoted in other currencies. A benefit of supporting multiple currencies is that as we add new asset classes from new countries, LEAN is already prepared to transact in those assets by using their quote currency. For instance, we added the India Equity market, which quotes assets in the INR currency.

The portfolio manages your currencies in its cashbook, which models the cash as a ledger of transactions. When you buy assets, LEAN uses the currencies in your cashbook to purchase the asset and pay the transaction fees. For more information about the cashbook, see Cashbook .

Buying Power

We model the margin requirements of each asset and reflect that in the buying power available to the algorithm. We source Futures margins from CME SPAN margins. Equity margin is 2x for standard margin accounts and 4x intraday for Pattern Day Trading accounts. For more information about buying power modeling, see Buying Power .

Examples

The following examples demonstrate some common practices for multi-asset modeling.

Example 1: BTC Spot-Crypto Future Arbitration

This algorithm demonstrates an arbitration between Spot and Crypto Future BTCUSDT using a BTC cash account. If one's price is above 0.5% of another's, we sell the relatively overpriced one and buy the counter side. To ensure the cash position is sufficient to open the buy position, we order the buy side after the sell side is filled and the cash is replenished.

```C#
public class MultiAssetModelingAlgorithm  : QCAlgorithm
{
    private Crypto _spot;
    private CryptoFuture _future;
    private decimal _threshold = 0.005m, _tradeQuantity;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Seed the last price to set the initial price of the BTCUSDT holdings.
        Settings.SeedInitialPrices = true;

        // Simulate a cash Bybit account.
        SetBrokerageModel(BrokerageName.Bybit, AccountType.Cash);
        SetAccountCurrency("USDT", 0);
        var spotBalance = 2;  // BTC
        SetCash("BTC", spotBalance);

        // Request BTCUSD spot and future data to trade their price discrepancies.
        _spot = AddCrypto("BTCUSDT", market: Market.Bybit);
        _future = AddCryptoFuture("BTCUSDT", market: Market.Bybit);

        // Simulate the portfolio is holding BTC cash initially via BTCUSDT position.
        // Note that the performance will also be affected by the BTC performance in the default account currency.
        _spot.Holdings.SetHoldings(averagePrice: _spot.Price, quantity: spotBalance);
        _tradeQuantity = spotBalance/2.0m;
    }

    public override void OnData(Slice slice)
    {
        // Wait for the Slice to contain QuoteBar objects for both assets.
        if (!(slice.QuoteBars.ContainsKey(_spot.Symbol) && slice.QuoteBars.ContainsKey(_future.Symbol)))
        {
            return;
        }
        // If the spot price is higher than the future price more than the threshold,
        // Do arbitration by selling the spot BTC and buying the future.
        if (_spot.Price >= _future.Price * (1m + _threshold) && !_future.Holdings.IsLong)
        {
            Sell(_spot.Symbol, _tradeQuantity);
        }
        // If the future price is higher than the spot price more than the threshold,
        // Do arbitration by buying the spot BTC and selling the future.
        if (_future.Price >= _spot.Price * (1m + _threshold) && _future.Holdings.IsLong)
        {
            Sell(_future.Symbol, _tradeQuantity);
        }
    }

    public override void OnOrderEvent(OrderEvent orderEvent)
    {
        // Order the buy-side of the arb only when the BTC is sold and USDT is obtained.
        if (!(orderEvent.Quantity < 0 && orderEvent.Status == OrderStatus.Filled))
        {
            return;
        }
        Security security = orderEvent.Symbol == _spot.Symbol ? _future : _spot;
        // Calculate the initial margin needed. We must sell the same amount of BTC to obtain sufficient USDT for
trade.
        var margin = security.BuyingPowerModel.GetInitialMarginRequirement(
            new InitialMarginParameters(security, _tradeQuantity)
        ).Value;
        // Check if USDT cash is sufficient to open the position.
        if (Portfolio.CashBook["USDT"].Amount >= margin)
        {
            Buy(security.Symbol, _tradeQuantity);
        }
    }
}
```

Example 2: BTC Spot-Future Arbitration

This algorithm implements a similar logic to the above example of an arbitration algorithm between spot BTCUSD and Future BTC using a cash account. If one's price is above 0.5% of another's, we sell the relatively overpriced one and buy the counter

side. Note that we need to calculate the future value of the Future fairly compared to the spot BTC. Also, we need to rollover any Futures contracts.

```C#
public class MultiAssetModelingAlgorithm  : QCAlgorithm
{
    private Crypto _spot;
    private Future _future;
    private decimal _threshold = 0.005m;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Set cash to sell Futures.
        SetCash(1000000);
        // Seed the price of each asset with its last known price to
        // avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Add BTCUSD spot and Future data to trade their price
        // discrepancies.
        _spot = AddCrypto("BTCUSD", market: Market.Coinbase);
        _future = AddFuture(Futures.Currencies.BTC, dataMappingMode: DataMappingMode.OpenInterest);
        _future.SetFilter(0, 62);
    }

    public override void OnData(Slice slice)
    {
        // Get the current bar for both markets.
        if (!slice.Bars.TryGetValue(_spot.Symbol, out var spot) || !slice.QuoteBars.TryGetValue(_future.Mapped, out
var future))
        {
            return;
        }
        // Use forward price to compare to spot BTC price fairly.
        var discountFactor = RiskFreeInterestRateModel.GetInterestRate(slice.Time) * (decimal)(_future.Mapped.ID.Date
- slice.Time).TotalSeconds / 60m / 60m / 24m / 365m;
        var btcFutureFV = future.Close * Convert.ToDecimal(Math.Exp((double)discountFactor));
        // If the spot price is above the Future price by more than the
        // threshold, sell the spot BTC and buy the Future to arbitrage.
        if (!_spot.Holdings.IsShort && spot.Close >= btcFutureFV * (1m + _threshold))
        {
            Sell(_spot.Symbol, _future.SymbolProperties.ContractMultiplier);
            Buy(_future.Mapped, 1m);
        }
        // If the Future price is above the spot price by more than the
        // threshold, sell the Future BTC and buy the spot BTC to arbitrage.
        else if (!_spot.Holdings.IsLong && btcFutureFV >= spot.Close * (1m + _threshold))
        {
            Sell(_future.Mapped, 1m);
            Buy(_spot.Symbol, _future.SymbolProperties.ContractMultiplier);
        }
        // When prices converge, exit the arbitrage trade.
        else if ((_spot.Holdings.IsLong && btcFutureFV <= spot.Close) || (_spot.Holdings.IsShort && spot.Close <=
btcFutureFV))
        {
            Liquidate();
        }
    }

    public override void OnSymbolChangedEvents(SymbolChangedEvents symbolChangedEvents)
    {
        // Get Symbol Change Event of the Continuous Future (change
        // in mapped contract) to roll over.
        if (!symbolChangedEvents.TryGetValue(_future.Symbol, out var changedEvent))
        {
            return;
        }
        var oldSymbol = changedEvent.OldSymbol;
        // Add the new contract.
        var newSymbol = AddFutureContract(changedEvent.NewSymbol).Symbol;
        var tag = $"Rollover - Symbol changed at {Time}: {oldSymbol} -> {newSymbol}";
        var quantity = Portfolio[oldSymbol].Quantity;
        // Rolling over: liquidate the position of the old mapped
        // contract and switch to the new mapped contract.
        Liquidate(oldSymbol, tag: tag);
        if (quantity != 0)
        {
            MarketOrder(newSymbol, quantity, tag: tag);
```

```
        }
    }
}
```

Key Concepts

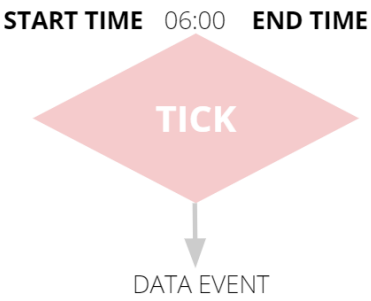
Time Modeling

Time Modeling

Periods

Introduction

Data comes in two different "shapes" according to the time period it covers: point values or period values. In QuantConnect, ticks are point values, and bars are a period values. These data formats have different properties, which control when LEAN emits them into your algorithm.



Start and End Time

Bars have a start and end time that represent the time period of which the bar aggregates data. LEAN passes the bar to your

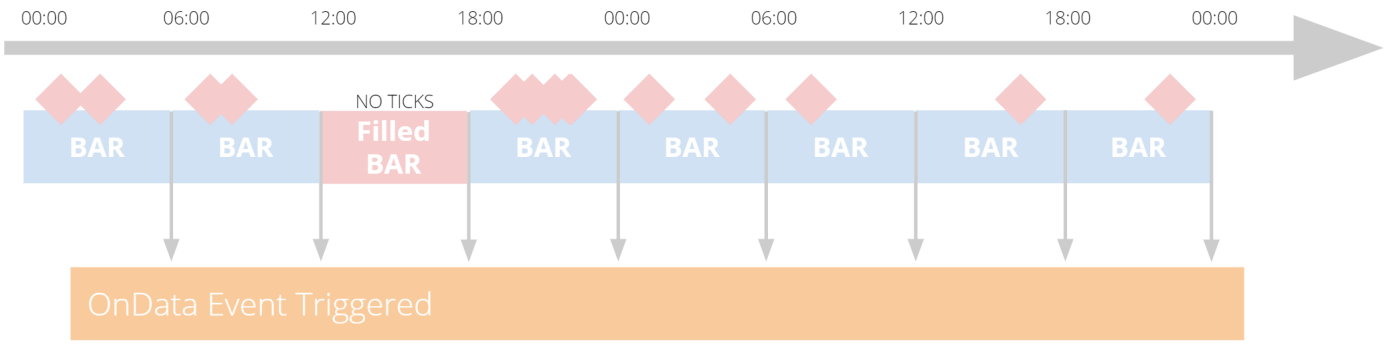
algorithm at the end time so that you don't receive the bar before it was actually available. Free online data providers commonly timestamp their bars to the start time of the bar and include the bar close price, making your research prone to look ahead bias.

Carefully consider the end-time of data to avoid one-off errors when you import the data into QC or compare indicator values across different platforms. Generally, bar timestamps can be represented as follows: `bar.EndTime = bar.Time + bar.Period`.

Period Values

Bars aggregate data across a period of time into a single object. We make this easy for you by pre-aggregating billions of raw trade-ticks into `TradeBar` objects and quote-ticks into `QuoteBar` objects.

We don't know the close of an intraday bar until the start of the next bar, which can sometimes be confusing. For daily bars, the bar will include all the ticks from market open to market close, but LEAN will emit it to your algorithm at market close. As a result, if you analyze the Friday data and then place an order, LEAN sends the order to your brokerage on Friday after the market close.



When there are no ticks during a period, LEAN emits the previous bar. This is the default behavior for bar data and it's referred to as "filling the data forward". You can enable and disable this setting when you create the security subscription.

We provide bar data in second, minute, hour, and daily bar formats. To create other periods of bars, see Consolidating Data.

Daily Periods

LEAN emits daily bars at market close or midnight, depending on your `DailyPreciseEndTime` setting. To be notified when a security has finished trading for the day, add an `OnEndOfDay` method to your algorithm. LEAN calls this method for each security every day.

```
C#
// Perform end-of-day logic for a given symbol.
public override void OnEndOfDay(Symbol symbol)
{

}
```

Point Values

Point values have no period because they occur at a singular point in time. Examples of point data includes ticks, open interest, and news releases. Tick data represents a single trade or quote in the market. It's a discrete event with no period, so the `Time` and `EndTime` properties are the same. LEAN emits ticks as soon as they arrive and doesn't fill them forward.

Examples

The following examples demonstrate some common practices for period time modeling.

Example 1: Tick Signals & Daily Bars

This example shows using a tick/pointwise signal from the Smart Insider Intention dataset while trading Starbuck's stock with daily bars. To liquidate the position after two trading days, you can count the number of `TradeBar` objects you get in the `OnData` method.

C#

```
public class PeriodTimeModelingAlgorithm : QCAlgorithm
{
    private Equity _equity;
    private Symbol _smartInsiderIntention;
    // Add a member to track the number of days until the position
    // should be liquidated.
    private int _holdDays = 0;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add an Equity asset. We will enter with a market on open order
        // and hold for 2 full days, so daily resolution is sufficient.
        _equity = AddEquity("SBUX", Resolution.Daily);
        // Add insider trade intention as the sentiment trading signal.
        _smartInsiderIntention = AddData<SmartInsiderIntention>(_equity.Symbol).Symbol;
    }

    public override void OnData(Slice slice)
    {
        // Buy Starbuck's stock whenever we receive a buyback intention
        // or transaction notification due to volatility.
        if (slice.ContainsKey(_smartInsiderIntention))
        {
            SetHoldings(_equity.Symbol, 1);
            // Hold the position for 2 days to fully digest the insider
            // intention sentiment.
            _holdDays = 2;
        }
        // Count down the holding day per each daily trade bar received.
        if (_equity.Holdings.Invested && slice.Bars.ContainsKey(_equity.Symbol) && _holdDays-- <= 0)
        {
            // LLiquidate the position if 2 days were fully counted.
            Liquidate(_equity.Symbol);
        }
    }
}
```

Time Modeling

Timeslices

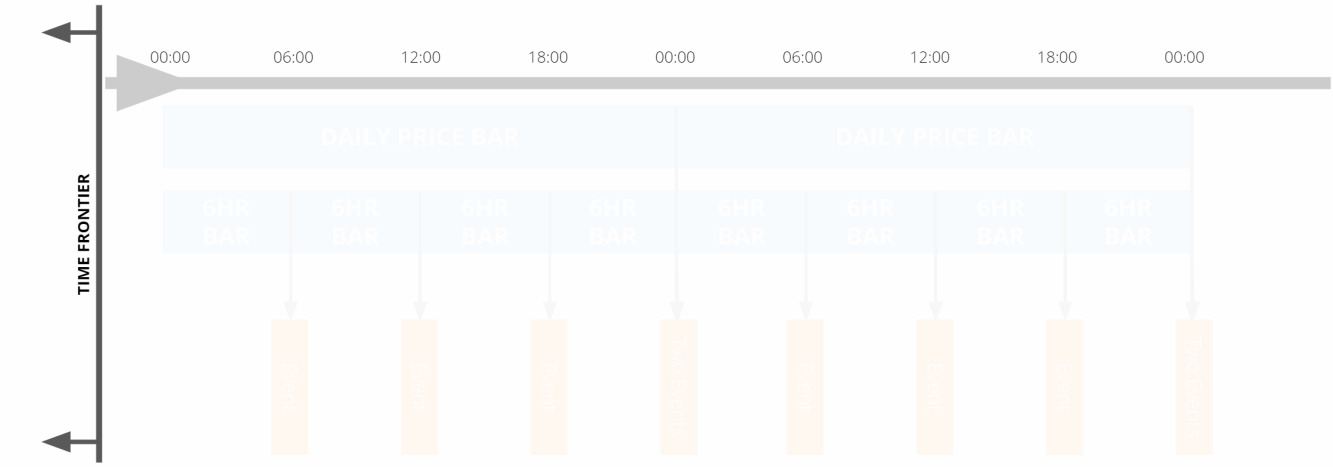
Introduction

The core technology behind QuantConnect algorithmic trading is an event-based, streaming analysis system called LEAN . LEAN attempts to model the stream of time as accurately as possible, presenting data ("events") to your algorithms in the order it arrives, as you would experience in reality.

All QuantConnect algorithms have this time-stream baked in as the primary event handler, `OnData` . The `Slice` object this method receives represents all of the data at a moment of time, a time-slice . No matter what data you request, you receive it in the order created according to simulated algorithm time. By only letting your algorithm see the present and past moments, we can prevent the most common quantitative-analysis error, look-ahead bias .

Time Frontier

If you request data for multiple securities and multiple resolutions, it can create a situation where one of your data subscriptions is ready to emit, but another subscription with a longer period is still be constructing its bar. Furthermore, if you request multiple datasets that have different time zones, your algorithm will receive the daily bars of each dataset at market close for the respective asset. To coordinate the data in these situations, we use the `EndTime` of each data point to signal when LEAN should transmit it to your algorithm.



Once your algorithm reaches the `EndTime` of a data point, LEAN sends the data to your `OnData` method. For intraday bars, this is the beginning of the next period. For daily bars, it's market close or midnight, depending on your `DailyPreciseEndTime` setting . To avoid look-ahead bias, you can only access data from before this Time Frontier. The `Time` property of your algorithm is always equal to the Time Frontier.

Properties

`Slice` objects have the following properties:

Get Time Slices

To get the current `Slice` object, define an `OnData` method or use the `CurrentSlice` property of your algorithm. The `Slice` contains all the data for a given moment in time. The `Bars bars` and `QuoteBars quote_bars` properties are `Symbol` /string indexed

dictionaries. The `Ticks ticks` property is a list of ticks for that moment of time, indexed by the `Symbol` . To check which data formats are available for each asset class, see the Data Formats page in the Asset Classes chapter.

The `Slice` object gives you the following ways to access your data:

- Indexing the `Slice` , which returns a dynamic object of your type.

  ```
  // Get the current dynamic object for the specified symbol by indexing the Slice object.
  public override void OnData(Slice slice)
  {
      var data = slice[_symbol];
  }
  ```

  With minute and second resolution data, the dynamic type is `TradeBar` for Equities and `QuoteBar` for other asset classes.

- Indexing the static properties, which returns the type you specify.

  ```
  // Get the current trade and quote bars for a specific symbol from the data slice.
  public override void OnData(Slice slice)
  {
      var tradeBars = slice.Bars[_symbol];
      var quoteBars = slice.QuoteBars[_symbol];
  }
  ```

- Calling the `Get<T>()` helper method.

  ```
  // Get trade bar, quote bar, and ticks for the specified symbol from the data slice.
  public override void OnData(Slice slice)
  {
      var tradeBar = slice.Get<TradeBar>(_symbol);
      var quoteBar = slice.Get<QuoteBar>(_symbol);
      var ticks = slice.Get<Ticks>(_symbol);
  }
  ```

Strongly typed access gives you compile-time safety, but dynamic type access can sometimes simplify coding. We recommend static types since they are easier to debug.

Check if the `Slice` contains the data you're looking for before you index it. If there is little trading, or you are in the same time loop as when you added the security, it may not have any data. Even if you enabled fill-forward for a security subscription, you should check if the data exists in the dictionary before you try to index it. To check if the `Slice` contains for a security, call the `ContainsKey` method. Note: if the `Slice` object doesn't contain any market data but it contains auxiliary data, the `slice.ContainsKey(symbol)` method can return true while `slice[symbol]` returns `null` .

```
// Check if the slice contains data for the symbol and retrieve it if it's available.
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_symbol))
    {
        var data = slice[_symbol];
    }
}
```

Examples

The following examples demonstrate some common practices for timeslices time modeling.

Example 1: 6-Hour Consolidated Bars

C#

```
public class TimeslicesTimeModelingAlgorithm : QCAlgorithm
{
    private Symbol _aapl;
    private TradeBarConsolidator _consolidator;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request AAPL data to trade it. We need a resolution denser than 6-hour for the consolidator.
        _aapl = AddEquity("AAPL", Resolution.Hour).Symbol;

        // Create a 6-hour consolidator for smoothing the noise.
        _consolidator = new TradeBarConsolidator(TimeSpan.FromHours(6));
        // Subscribe the consolidator to update with the security's data automatically.
        SubscriptionManager.AddConsolidator(_aapl, _consolidator);
    }

    public override void OnData(Slice slice)
    {
        if (slice.Bars.TryGetValue(_aapl, out var bar) && _consolidator.WorkingBar != null)
        {
            // Trade on a rising trend, suggested by the current close is above the past hour open
            // and the past hour open is above the 6-hour bar open.
            if (_consolidator.WorkingBar.Open < bar.Open && bar.Open < bar.Close)
            {
                SetHoldings(_aapl, 1m);
            }
            // Trade on a down trend, suggested by the current close is below the past hour open
            // and the past hour open is below the 6-hour bar open.
            else if (_consolidator.WorkingBar.Open > bar.Open && bar.Open > bar.Close)
            {
                SetHoldings(_aapl, -1m);
            }
            // Otherwise, do not hold a position if there is no deterministic trend.
            else
            {
                Liquidate();
            }
        }
    }
}
```

Time Modeling

Time Zones

Introduction

LEAN is an international trading engine, so it can operate across time zones. The exchange, your algorithm, and the data in your algorithm can all have different time zones.

Exchange Time Zone

Exchanges are located in different areas, so they can have different time zones. To get the time zone of the exchange that a `Security` trades on, use its `Exchange.Hours.TimeZone` property.

```
C#

// Get the time zone of the exchange for a specific asset.
var timeZone = Securities[_symbol].Exchange.Hours.TimeZone;
```

Algorithm Time Zone

LEAN supports international trading across multiple time zones and markets, so it needs a reference time zone for the algorithm to set the `Time` . The default time zone is Eastern Time (ET), which is UTC-4 in summer and UTC-5 in winter. To set a different time zone, call the `SetTimeZone` method. This method accepts either a string following the IANA Time Zone database convention or a NodaTime .DateTimeZone object. If you pass a string, the method converts it to a `NodaTime.DateTimeZone` object. The `TimeZones` class provides the following helper attributes to create `NodaTime.DateTimeZone` objects:

```
C#

SetTimeZone("Europe/London");
SetTimeZone(NodaTime.DateTimeZone.Utc);
SetTimeZone(TimeZones.Chicago);
```

To get the time zone of your algorithm, use the `TimeZone` property.

```
C#

// The algorithm timezone property can assist with multi-asset trading.
var timeZone = TimeZone;
```

To get the algorithm time in Coordinated Universal Time (UTC), use the `UtcTime` property.

```
C#

// Access the current UTC time to coordinate multi-timezone events or improve logging.
var utcTime = UtcTime;
```

The `Time` and `UtcTime` objects have no time zone. LEAN maintains their state to be consistent.

Data Time Zone

Datasets can have different time zones because the time zone is up to the data provider, where they are located, and where they collect the data. LEAN tracks the time zone of each dataset so that you receive the correct data at the right point in time in your algorithm. If you have multiple datasets in your algorithm from different time zones, LEAN synchronizes them to your

algorithm time. Furthermore, if you set up time period consolidators , LEAN consolidates the data based on the data time zone.

To get the time zone of a dataset, view the listing page in the Dataset Market or call the `GetDataTimeZone` method.

```C#
// Get the timezone of an asset.
var timeZone = MarketHoursDatabase.GetDataTimeZone(Market.USA, _symbol, SecurityType.Equity);
```

Examples

The following examples demonstrate some common practices for time zone modeling.

Example 1: Futures Time Zone

The following algorithm changes its time zone to be the same as the EMini SP500 Futures exchange, i.e., Chicago time zone. It is comparable in validating data across different sources and has a more convenient time notification without further conversion.

C#

```
public class TimeZoneModelingAlgorithm : QCAlgorithm
{
    private Future _future;
    private OrderTicket _ticket;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request ES future data for trading.
        _future = AddFuture("ES", Resolution.Minute,
            extendedMarketHours: true,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            dataMappingMode: DataMappingMode.OpenInterest,
            contractDepthOffset: 0);

        // Set the time zone to be the same as the Future (Chicago).
        // It will make the time in the backtest more comparable and easily analyzed with any other data sources.
        // Performance of the backtest will only be altered by this change if the trading logic depends on the Time
property.
        SetTimeZone(_future.Exchange.TimeZone);
    }

    public override void OnData(Slice slice)
    {
        if (!Portfolio.Invested && _ticket == null)
        {
            var contract = Securities[_future.Mapped];
            // Set a limit price that repsects the brokerage's precision requirements.
            var limitPrice = AdjustLimitPrice(contract, contract.Price * 0.995m);
            _ticket = LimitOrder(contract.Symbol, 1, limitPrice);
        }

        // Cancel unfilled orders if they are not filled within 2 minutes to avoid stale fills.
        // Note that the Time property in the order ticket is in the UTC timezone.
        if (_ticket.Status != OrderStatus.Filled && _ticket.Time.AddMinutes(2) < UtcTime)
        {
            // You can change to a different time zone and compare the log differences.
            _ticket.Cancel($"{Time} :: utc: {UtcTime} :: Order canceled due to placed over 2 minutes");

            _ticket = null;
        }
    }

    private decimal AdjustLimitPrice(Security security, decimal price, bool roundUp = true)
    {
        decimal pip = security.PriceVariationModel.GetMinimumPriceVariation(
            new GetMinimumPriceVariationParameters(security, price)
        );
        return ((int)(price / pip) + (roundUp ? 1 : 0)) * pip;
    }
}
```

Key Concepts

Security Identifiers

Introduction

We implemented `Symbol` objects in the LEAN open-source project as a way to identify or "finger-print" tradable assets so that no further database look-up is required. All QuantConnect and LEAN Algorithm API methods use `Symbol` objects to identify assets.

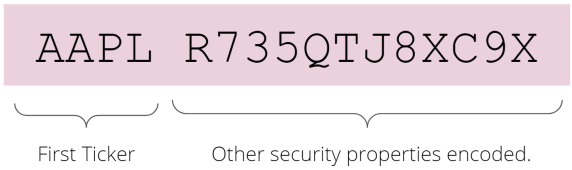
Encoding Symbols

`SecurityIdentifier` objects have several public properties to uniquely identify each asset. The following table describes these properties:

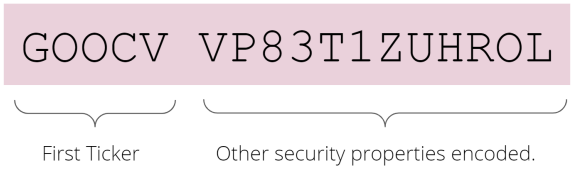
Property	Data Type	Description
`Market`	`string`	The market in which the asset trades
`SecurityType`	`SecurityType`	The asset class
`OptionStyle`	`OptionStyle`	American or European Option style
`OptionRight`	`OptionRight`	Call or put Option type
`StrikePrice`	`decimal`	The Option contract strike price
`Date`	`DateTime`	Earliest listing date for Equities; expiry for Futures and Options; December 30, 1899 for continuous Futures contracts and Indices.
`HasUnderlying`	`bool`	A flag to represent if its a derivative asset with another underlying asset

We encode the preceding properties to create `Symbol` objects. We do our best to hide the details of this process from your algorithm, but you'll occasionally see it come through as an encoded hash like `AAPL R735QTJ8XC9X` . The first half of the encoded string represents the first ticker AAPL was listed under. The other letters at the end of the string represent the serialized form of the preceding `SecurityIdentifier` properties. You may also see an encoded has like `AAPL XL7X5I241SO6|AAPL R735QTJ8XC9X` for a derivative contract. In this case, the part before the `|` is contract hash and the part after the `|` is the underlying hash. This serialization approach lets LEAN assign a short, unique string to octillions of different security objects.

Apple Inc. (same name since listed).



Google Class-C Shares (renamed to GOOG).



LEAN assumes the ticker you pass to the AddEquity method is the current ticker of the Equity asset. To access this ticker, use the `Value` property of the `Symbol` object.

```
C#

// Add equity for "GOOG" and print symbol id value, symbol value and encoded symbol id.
var symbol = AddEquity("GOOG").Symbol;
Debug(symbol.ID.Value);       // Prints "GOOCV"
Debug(symbol.Value);          // Prints "GOOG"
Debug(symbol.ID.ToString());  // Prints "GOOCV VP83T1ZUHROL"
```

If you create the security subscription with a universe selection function, the `Value` property of the `Symbol` object is the point-in-time ticker.

Decoding Symbols

When a `SecurityIdentifier` is serialized to a string, it looks something like `SPY R735QTJ8XC9X` . This two-part string is a base64 encoded set of data. Encoding all of the properties into a short format allows dense communication without requiring a third-party list or look-up. Most of the time, you will not need to work with these encoded strings. However, to deserialize the string into a `Symbol` object, call the `Symbol` method.

```
C#

// Extract the market, security type, and value from the symbol with ID "GOOCV VP83T1ZUHROL".
var symbol = Symbol("GOOCV VP83T1ZUHROL");
symbol.ID.Market;     // => "USA"
symbol.SecurityType;  // => SecurityType.Equity
symbol.Value;         // => "GOOCV"
```

Tickers

Tickers are a string shortcode representation for an asset. Some examples of popular tickers include "AAPL" for Apple Corporation and "IBM" for International Business Machines Corporation. These tickers often change when the company rebrands or they undergo a merger or reverse merger.

The ticker of an asset is not the same as the `Symbol` . `Symbol` objects are permanent and track the underlying entity. When a company rebrands or changes its name, the `Symbol` object remains constant, giving algorithms a way to reliably track assets over time.

Tickers are also often reused by different brokerages. For example Coinbase, a leading US Crypto Brokerage lists the "BTCUSD" ticker for trading. Its competitor, Bitfinex, also lists "BTCUSD". You can trade both tickers with LEAN. `Symbol` objects allow LEAN to identify which market you reference in your algorithms.

To create a `Symbol` object for a point-in-time ticker, call the `GenerateEquity` method to create the security identifier and then

call the `Symbol` constructor. For example, Heliogen, Inc. changed their ticker from ATHN to HLGN on December 31, 2021. To convert the ATHN ticker to the Equity `Symbol` , type:

```
C#
// Generate an Equity security identifier for a ticker and then create a symbol with it.
var ticker = "ATHN";
var securityID = SecurityIdentifier.GenerateEquity(ticker, Market.USA, mappingResolveDate:new DateTime(2021, 12, 1));
var symbol = new Symbol(securityID, ticker);
```

In the preceding code snippet, the `mappingResolveDate` must be a date when the point-in-time ticker was trading.

Symbol Cache

To make using the API easier, we have built the Symbol Cache technology, which accepts strings and tries to guess which `Symbol` object you might mean. With the Symbol Cache, many methods can accept a string such as "IBM" instead of a complete `Symbol` object. We highly recommend you don't rely on this technology and instead save explicit references to your securities when you need them.

```
C#
// Example 1: Relying On Symbol Cache:
AddEquity("IBM");          // Add by IBM string ticker, save reference to Symbol Cache.
MarketOrder("IBM", 100);   // Determine refering to IBM Equity from Symbol Cache.
History("AAPL", 14);       // Guess referring to AAPL Equity.

// Example 2: Correctly Using Symbols:
var ibm = AddEquity("IBM").Symbol;   // Add IBM Equity string ticker, save Symbol.
MarketOrder(ibm, 100);               // Use IBM Symbol in future API calls.

var aapl = Symbol.Create("AAPL", SecurityType.Equity, Market.USA);
History(aapl, 14);
```

Markets

The `Market` property distinguishes between tickers that have the same string value but represent different underlying assets. A prime example of this is the various market makers who have different prices for EURUSD. We store this data separately and as they have different fill prices, we treat the execution venues as different markets.

The following table shows the available markets for each security type:

Security Type	Market(s)
SecurityType.Equity	Market.USA
SecurityType.Option	Market.USA
SecurityType.Crypto	Market.Binance Market.BinanceUS Market.Bitfinex Market.Bybit Market.Coinbase Market.Kraken
SecurityType.CryptoFuture	Market.Binance Market.Bybit Market.dYdX
SecurityType.Forex	Market.Oanda
SecurityType.Future	Market.CBOT Market.CFE Market.CME Market.COMEX Market.EUREX Market.HKFE Market.ICE Market.India Market.NYMEX Market.NYSELIFFE Market.SGX
SecurityType.FutureOption	Market.CBOT Market.COMEX Market.NYSELIFFE
SecurityType.Index	Market.EUREX Market.HKFE Market.OSE Market.USA
SecurityType.IndexOption	Market.USA
SecurityType.Cfd	Market.InteractiveBrokers Market.Oanda

Industry Standard Identifiers

You can convert industry-standard security identifiers like CUSIP, FIGI, ISIN, and SEDOL to `Symbol` objects and you can convert `Symbol` objects to industry-standard security identifiers.

CUSIP

To convert a Committee on Uniform Securities Identification Procedures (CUSIP) number to a `Symbol` or a `Symbol` to a CUSIP number, call the `CUSIP` method.

```C#
// Get the CUSIP for the symbol and vice versa.
var symbol = CUSIP("03783310"); // AAPL Symbol
var cusip = CUSIP(symbol); // AAPL CUSIP (03783310)
```

FIGI

To convert a Financial Instrument Global Identifier (FIGI) to a `Symbol` or a `Symbol` to a FIGI, call the `CompositeFIGI` method.

```
// Get the FIGI for the symbol and vice versa.
var symbol = CompositeFIGI("BBG000B9XRY4"); // AAPL Symbol
var figi = CompositeFIGI(symbol); // AAPL FIGI (BBG000B9XRY4)
```

ISIN

To convert an International Securities Identification Number (ISIN) to a `Symbol` or a `Symbol` to an ISIN, call the `ISIN` method.

```
// Get the ISIN for the symbol and vice versa.
var symbol = ISIN("US0378331005"); // AAPL Symbol
var isin = ISIN(symbol); // AAPL ISIN (US0378331005)
```

SEDOL

To convert a Stock Exchange Daily Official List (SEDOL) number to a `Symbol` or a `Symbol` to a SEDOL number, call the `SEDOL` method.

```
// Get the SEDOL for the symbol and vice versa
var symbol = SEDOL("2046251"); // AAPL Symbol
var sedol = SEDOL(symbol); // AAPL SEDOL (2046251)
```

Examples

The following examples demonstrate some common practices for using the security identifier.

Example 1: Future Contract Expiry

The following example demonstrates the use of the security identifier of a Future contract to access its expiry date for contract selection and schedule a singular non-repeating event to rollover using `future.Symbol.ID.Date` .

C#

```
public class SecurityIdentifierExampleAlgorithm : QCAlgorithm
{
    private Future _future;
    private OrderTicket _ticket;
    private ScheduledEvent _lastScheduledEvent;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Seed the price data to allow the mapped contract to fill despite sparse data.
        Settings.SeedInitialPrices = true;
    }

    public override void OnWarmupFinished()
    {
        // Since we do not subscribe to any data source, we can use the OnWarmupFinished event handler instead of
OnData for the initial position.
        var es = QuantConnect.Symbol.Create(Futures.Indices.SP500EMini, SecurityType.Future, Market.CME);
        OpenFuturePosition(es, Time.AddDays(30), 1);
    }

    private void OpenFuturePosition(Symbol symbol, DateTime minExpiry, decimal quantity)
    {
        // Select the next mapped future contract with the nearest expiry,
        // which should be further than the current contract.
        var selected = FuturesChain(symbol).Where(x => x.ID.Date > minExpiry).First();
        // Subscribe to its data to trade it.
        _future = AddFutureContract(selected, extendedMarketHours: true);

        // Buy back the corresponding number of contracts as the previous mapped contract.
        MarketOrder(_future.Symbol, quantity);

        // Schedule the next rollover to be 1 week before the expiry to avoid excessive volatility.
        // It will only be triggered once 7 days before the expiry and discarded afterward.
        if (_lastScheduledEvent != null) Schedule.Remove(_lastScheduledEvent);
        Schedule.On(
            DateRules.On(new[] { _future.Symbol.ID.Date.AddDays(-7) }),
            TimeRules.At(9, 30),
            RollOver
        );
    }

    private void RollOver()
    {
        // Save the current number of contracts to roll over to the next mapped contract.
        var quantity = _future.Holdings.Quantity;
        // Liquidate the to-be-expired contract to roll over the position.
        Liquidate(_future.Symbol);

        OpenFuturePosition(_future.Symbol.Canonical, _future.Symbol.ID.Date, quantity);
    }
}
```

Key Concepts

Event Handlers

Introduction

Event handlers are called during an algorithm execution to pass information to your strategy. Most of the events are not needed for simple strategies, but can be helpful for debugging issues in more complex algorithms.

Data Events

```C#
// Get data for a specific symbol to analyze its price movements or trading signals.
public override void OnData(Slice slice)
{
    slice.TryGetValue(_symbol, out var myData);
}

// Get split information to adjust holdings and calculations for the asset.
public override void OnSplits(Splits splits)
{
    splits.TryGetValue(_symbol, out var split);
}

// Capture dividend events to account for income or adjust the strategy based on the dividend payout.
public override void OnDividends(Dividends dividends)
{
    dividends.TryGetValue(_symbol, out var dividend);
}

// Track events when a security changes its ticker, allowing the algorithm to adapt to these changes.
public override void OnSymbolChangedEvents(SymbolChangedEvents symbolChangedEvents)
{
    symbolChangedEvents.TryGetValue(_symbol, out var symbolChangedEvent);
}

// Handle delisting events to manage the position or adjust the strategy when the symbol is delisted.
public override void OnDelistings(Delistings delistings)
{
    delistings.TryGetValue(_symbol, out var delisting);
}
```

The `OnData` method is the primary event handler for receiving financial data events to your algorithm. It is triggered sequentially at the point in time the data is available; in backtesting and live. For daily data, this means the event is processed at market close. In this method, all data for a given moment of time is grouped in a single event, including custom data types. This data is passed with the Slice object.

When fill-forward is enabled for your asset, the OnData event handler will be called regularly even if there was no new data. This is the default behavior.

In backtesting, if your algorithm takes a long time to process a slice, the following slice objects queue up and the next event triggers when your algorithm finishes processing the current slice. In live trading, if your algorithm takes longer to process the current slice than the time period that the slice spans, the next event triggers when your algorithm finishes processing the current slice, but the slice of the following event is the most recent slice. For example, say your algorithm consumes second resolution Crypto data but it takes your algorithm 3.5 seconds to process each slice. In backtesting, you'll get every slice. In live trading, if you deploy at 12:00:00 AM Coordinated Universal Time (UTC), you'll get the first slice at 12:00:01 AM UTC (spanning 12:00:00 AM UTC to 12:00:01 AM UTC) and you'll get the second slice at 12:00:04.5 AM UTC (roughly spanning 12:00:03 AM UTC to 12:00:04 AM UTC).

The `OnSplits` , `OnDividends` , `OnSymbolChangedEvents` , and `OnDelistings` event handlers provide data for their respective types in an isolated way. However, all of the data for these corporate actions is also available in the `Slice` in `OnData` .

To perform intensive computation before the market opens, use a Scheduled Event or the Train method.

For more information on the `Slice` object and OnData event, see Handling Data .

Securities Changed Event

```
C#
// Log the universe changes.
public override void OnSecuritiesChanged(SecurityChanges changes)
{
    foreach (var security in changes.AddedSecurities)
    {
        Debug($"{Time}: Added {security.Symbol}");
    }
    foreach (var security in changes.RemovedSecurities)
    {
        Debug($"{Time}: Removed {security.Symbol}");

        if (security.Invested)
        {
            Liquidate(security.Symbol, "Removed from Universe");
        }
    }
}
```

The `OnSecuritiesChanged` event notifies the algorithm when assets are added or removed from the universe. This can be due to changes in the Universe constituents , delisting , contract expiration, or an explicit call to the RemoveSecurity method.

The event is triggered immediately when the asset is removed from the universe; however the data feed for the asset may remain active if the algorithm has open orders.

For more information, see how to use Security Changed Events .

Order Events

```
C#
// Track filled orders and get the details of each fill.
public override void OnOrderEvent(OrderEvent orderEvent)
{
    var order = Transactions.GetOrderById(orderEvent.OrderId);
    if (orderEvent.Status == OrderStatus.Filled)
    {
        Debug($"{Time}: {order.Type}: {orderEvent}");
    }
}

// Track Option assignment events to handle assigned Equity holdings.
public override void OnAssignmentOrderEvent(OrderEvent assignmentEvent)
{
    Log(assignmentEvent.ToString());
}
```

The `OnOrderEvent` method notifies the algorithm of new orders, and changes in the order status such as fill events and cancelations. For options assignment events there is a dedicated event handler `OnAssignmentOrderEvent` .

In backtesting order events are triggered synchronously after the main data events. In live trading, order events are asynchronously as they occur. To avoid infinite loops, we recommend not to place orders in the OnOrderEvent.

For more information, see how to use Order Events .

Brokerage Events

```
// Track brokerage messages.
public override void OnBrokerageMessage(BrokerageMessageEvent message)
{
    if (message.Type== BrokerageMessageType.Reconnect)
    {
        Log($"{Time}: {message.Type}: Message: {message.Message}");
    }
}

// Track brokerage disconnections and respond to connectivity interruptions.
public override void OnBrokerageDisconnect()
{
    Log($"Brokerage disconnection detected at {Time}.");
}

// Track brokerage reconnections to resume trading operations.
public override void OnBrokerageReconnect()
{
    Log($"Brokerage reconnected at {Time}.");
}
```

The `OnBrokerageDisconnect` and `OnBrokerageReconnect` event handlers pass information to the algorithm about the brokerage connection status. This can be helpful for considering when to place a trade when a brokerage API is unreliable or under maintenance. The `OnBrokerageMessage` event handler provides information from the brokerages, including the disconnect and reconnect messages. Message content varies with each brokerage.

Brokerage events are triggered asynchronously in live trading, and are not created in backtesting.

Margin Call Events

```
// Track when remaining margin is low.
public override void OnMarginCallWarning()
{
    Log("Margin call warning, 5% margin remaining");
}

// Review and adjust liquidation orders in response to a margin call.
public override void OnMarginCall(List<SubmitOrderRequest> requests)
{
    // Modify order requests to choose what to liquidate.
    foreach (var orderRequest in requests)
    {
    }
}
```

The `OnMarginCallWarning` and `OnMarginCall` event handlers provide information and control over how to reduce algorithm leverage when performance is poor.

The `OnMarginCallWarning` method is called when there is less than 5% margin remaining to give you an opportunity to reduce leverage before an official margin call is performed by the brokerage. The `OnMarginCall` event handler is passed a list of `SubmitOrderRequest` objects which will be executed on exiting the method.

Margin events are called before the data events are processed.

The following demonstration processes suggested orders and modifies the qualities to liquidate more than necessary and prevent a second margin call. For more information, see how to handle Margin Calls .

Warmup Finished Event

```
// This event handler runs when the algorithm is done warming up.
public override void OnWarmupFinished()
{
    // Done warming up.
}
```

The `OnWarmupFinished` event handler runs once initialization and warm up is complete.

Command Events

```
// An event handler for receiving external commands.
public override bool? OnCommand(dynamic data)
{
    Log($"Got command at {Time} with data: {data}");
    return true;
}
```

The `OnCommand` event handler runs when you send external commands to your algorithm.

End Of Algorithm Events

```
// The final step after algorithm execution.
public override void OnEndOfAlgorithm()
{
    Debug("Algorithm done");
}
```

The `OnEndOfAlgorithm` event handler is when the algorithm has finished executing as its final step. This event handler is helpful for performing final analysis, or saving algorithm state.

Event Flow

When you deploy an algorithm, LEAN first calls the Initialize method. In live mode, the engine loads your holdings and open orders from your brokerage account to add data subscriptions and populate the `Securities` , `Portfolio` , and `Transactions` objects. LEAN receives the data from the subscriptions, synchronizes the data to create a timeslice , and then performs the following steps. Your algorithm can spend up to 10 minutes on each timeslice unless you call the Train method.

1. If it's a backtest, check if there are Scheduled Events in the past that didn't fire because there was no data between the previous slice and the current slice. LEAN automatically creates a Scheduled Event to call the `OnEndOfDay` method at the end of each day.

   In live mode, Scheduled Events occur in a separate thread from the algorithm manager, so they run at the correct time.

2. Update the algorithm time.
3. Update the `CurrentSlice` .
4. Pass the `SymbolChangedEvents` to the `OnSymbolChangedEvents` method.
5. Cancel all open orders for securities that changed their ticker .
6. Add a `Security` object to the `Securities` collection for each new security in the universe.
7. Update the `Security` objects with the latest data.
8. Update the `Cash` objects in the CashBook with the latest data.
9. Process fill models for non-market orders.
10. Submit market on open orders to liquidate Equity Option contracts if the underlying Equity has a split warning .

11. Process margin calls .
12. If it's time to settle unsettled cash , perform settlement.
13. Call the OnSecuritiesChanged method with the latest security changes.
14. Apply dividends to the portfolio.
15. For securities that have a split warning, update their portfolio holdings and adjust their open orders to account for the split.
16. Update consolidators with the latest data.
17. Pass custom data to the `OnData` method.
18. Pass the `Dividends` to the `OnDividends` method.
19. Pass the `Splits` to the `OnSplits` method.
20. Pass the `Delistings` to the `OnDelistings` method.
21. Pass the `Slice` to the `OnData` method.
22. Perform universe selection .
23. Pass the `Slice` to the `Update` method of each Alpha model .
24. Pass the `Insight` objects from the Alpha model to the Portfolio Construction model .
25. Pass the `PortfolioTarget` objects from the Portfolio Construction model to each Risk Management model .
26. Pass the risk-adjusted `PortfolioTarget` objects from the Risk Management models to the Execution model .

When your algorithm stops executing, LEAN calls the `OnEndOfAlgorithm` method .

Key Concepts

Python and LEAN

Introduction

The LEAN open-source package powers the core functionality of QuantConnect. Strategies plugin to LEAN to request data, and process trades. LEAN is written in C# and accessed in python with a defined API. Most of these interactions are using native python types, with a few key exceptions. To comfortably use LEAN you should learn to recognize error messages when you hit a non-native API, and how to convert back to a native type.

Enumerable Types

When LEAN returns an enumerable, the object should be converted to a `list()` to be easily read by python classes. Popular API methods which return an enumerable are shown below:

Historical Data

QuantConnect also provides methods to request historical data that return a pandas dataframe, or an enumerable. If loading a large amount of data using an enumerable streams the results reducing RAM usage.

The enumerable types might look like the `MemoizingEnumerable` that is a raw C# type accessible in python.

```
qb = QuantBook()
start = datetime(2023, 2, 1)
end = datetime(2023, 2, 4)
ibm = qb.AddEquity('IBM', Resolution.Minute)
bars = qb.History[TradeBar](ibm.Symbol, start, end, Resolution.Minute)
bars
```

✓ 0.3s Python

```
<QuantConnect.Util.MemoizingEnumerable[TradeBar] object at 0x7fd5313209c0>
```

You can directly enumerate this object with standard python `for` loops, or to convert it to a native type, wrap the object in the `list()`:

```
# Loop over enumerable
import numpy as np
close = []
for bar in bars:
    close.append( bar.Close )
print( np.mean(close) )

# Manipulate as a list
barList =  list(bars)
len(barList)
```

✓ 0.8s Python

```
135.29654786324787

1170
```

Underlying Plugin Architecture

Your strategy code implements the QCAlgorithm class, which can communicate with the LEAN Engine. These method calls are done at the same speed as cython, directly invoking calls at the C layer of python. Your algorithm is imported as syntax checked, interpreted python.

Key Concepts

Duck Typing

Introduction

Duck typing is popular type system in Python. The language documentation defines duck typing as follows:

> A programming style which does not look at an object's type to determine if it has the right interface; instead, the method or attribute is simply called or used ("If it looks like a duck and quacks like a duck, it must be a duck.") By emphasizing interfaces rather than specific types, well-designed code improves its flexibility by allowing polymorphic substitution.
>
> Duck-typing avoids tests using `type()` or `isinstance()` . (Note, however, that duck-typing can be complemented with abstract base classes .) Instead, it typically employs `hasattr()` tests or EAFP programming.

You can't use duck-typing to add members to regular C# types.

```C#
// Duck-typing doesn't work on regular C# types.
class Duck { }
var duck = new Duck();
duck.quacks = "Quack!"   // This yields a compilation error.

dynamic duck = new Duck();
duck.quacks = "Quack!"   // This yields a runtime error.
```

LEAN allows duck-typing in the `Security` type.

Custom Security Properties

You can add properties to the `Security` object. For example, you can add an exponential moving average .

```C#
// Cast the Equity to a dynamic object to add an EMA indicator to it.
dynamic equity = AddEquity("SPY");
equity.ema = EMA(equity.Symbol, 10, Resolution.Daily);
```

This feature is helpful because you can get the `Security` object from the `Securities` object.

```C#
// Get the SPY Equity object from Securities to get the EMA indicator.
var ema = (decimal)(Securities["SPY"] as dynamic).ema.Current.Value;
```

To avoid casting to and from the `dynamic` type, you can use the `Set` method to assign and the `Get<T>` method to access the dynamic member and return the object with its type.

```
var equity = AddEquity("SPY");
// Assign an EMA indicator to the "ema" property
equity.Set("ema", EMA(equity.Symbol, 10, Resolution.Daily));
// Get the SPY Equity object from Securities to get the EMA indicator and its current value
var ema = Securities["SPY"].Get<ExponentialMovingAverage>("ema");   // The type of ema is ExponentialMovingAverage
var emaValue = ema.Current.Value;    // The type of emaValue is decimal
```

If the type or the member name is incorrect, the `Get<T>` method causes a runtime error.

Other Types

You can only add properties to the `Security` type.

Examples

The following examples demonstrate some common practices for duck typing.

Example 1: Store EMA Indicator

The following example implements a trend-following strategy using the EMA indicator for the top 20 liquid equities. It uses duck typing to store the EMA indicator in each Security object , so the indicator is discarded together with the `Security` when it is removed from the universe.

C#

```
public class DuckTypingAlgorithm : QCAlgorithm
{
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Weekly renewal on the universe to allow time to capitalize on the trend.
        UniverseSettings.Schedule.On(DateRules.WeekStart());
        // Trade on the top 20 liquid equities since they have higher volume and price movements.
        _universe = AddUniverse(Universe.Top(20));
    }

    public override void OnData(Slice slice)
    {
        foreach (var (symbol, security) in _universe.Members)
        {
            // Trade on updated price data.
            if (slice.Bars.TryGetValue(symbol, out var bar))
            {
                var ema = (security as dynamic).ema;
                // Switch to long if the current price is above EMA, suggesting an uptrend.
                if (bar.Close > ema && !Portfolio[symbol].IsLong)
                {
                    SetHoldings(symbol, 0.025m);
                }
                // Switch to short if the current price is below EMA, suggesting a downtrend.
                else if (bar.Close < ema && !Portfolio[symbol].IsShort)
                {
                    SetHoldings(symbol, -0.025m);
                }
            }
        }
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach (var removed in changes.RemovedSecurities)
        {
            // Liquidate if it is no longer in the top 50 liquidity stocks.
            Liquidate(removed.Symbol);
            // Release resources that update automatic indicators
            DeregisterIndicator(removed.Get<ExponentialMovingAverage>("ema"));
        }

        foreach (var added in changes.AddedSecurities)
        {
            // Use duck typing to create an EMA indicator for trend trading.
            // Discarded together when the security is removed from the universe, you won't need to handle it.
            var security = added as dynamic;
            security.ema = EMA(added.Symbol, 60, Resolution.Daily);
            // Warm up the EMA indicator to ensure its readiness for immediate usage.
            WarmUpIndicator(added.Symbol, (ExponentialMovingAverage)security.ema, Resolution.Daily);
        }
    }
}
```

Key Concepts

Research Guide

Introduction

We aim to teach and inspire our community to create high-performing algorithmic trading strategies. We measure our success by the profits community members create through their live trading. As such, we try to build the best quantitative research techniques possible into the product to encourage a robust research process.

Hypothesis-Driven Research

We recommend you develop an algorithmic trading strategy based on a central hypothesis. You should develop an algorithm hypothesis at the start of your research and spend the remaining time exploring how to test your theory. If you find yourself deviating from your core theory or introducing code that isn't based around that hypothesis, you should stop and go back to thesis development.

Wang et al. (2014) illustrate the danger of creating your hypothesis based on test results. In their research, they examined the earnings yield factor in the technology sector over time. During 1998-1999, before the tech bubble burst, the factor was unprofitable. If you saw the results and then decided to bet against the factor during 2000-2002, you would have lost a lot of money because the factor performed extremely well during that time.

Hypothesis development is somewhat of an art and requires creativity and great observation skills. It is one of the most powerful skills a quant can learn. We recommend that an algorithm hypothesis follow the pattern of cause and effect. Your aim should be to express your strategy in the following sentence:

> *A change in {cause} leads to an {effect}.*

To search for inspiration, consider causes from your own experience, intuition, or the media. Generally, causes of financial market movements fall into the following categories:

- Human psychology
- Real-world events/fundamentals
- Invisible financial actions

Consider the following examples:

Cause	*leads to*	Effect
Share class stocks are the same company, so any price divergence is irrational...		A perfect pairs trade. Since they are the same company, the price will revert.
New stock addition to the S&P500 Index causes fund managers to buy up stock...		An increase in the price of the new asset in the universe from buying pressure.
Increase in sunshine-hours increases the production of oranges...		An increase in the supply of oranges, decreasing the price of Orange Juice Futures.
Allegations of fraud by the CEO causes investor faith in the stock to fall...		A collapse of stock prices for the company as people panic.
FDA approval of a new drug opens up new markets for the pharmaceutical company...		A jump in stock prices for the company.
Increasing federal interest rates restrict lending from banks, raising interest rates...		Restricted REIT leverage and lower REIT ETF returns.

There are millions of potential alpha strategies to explore, each of them a candidate for an algorithm. Once you have chosen a strategy, we recommend exploring it for no more than 8-32 hours, depending on your coding ability.

Data Mining Driven Research

An alternative view is to follow any statistical anomaly without explaining it. In this case, you can use statistical techniques to identify the discontinuities and eliminate them when their edge is gone. Apparently, Renaissance Technologies has data mining models like this.

Overfitting

Overfitting occurs when you fine-tune the parameters of an algorithm to fit the detail and noise of backtesting data to the extent that it negatively impacts the performance of the algorithm on new data. The problem is that the parameters don't necessarily apply to new data and thus negatively impact the algorithm's ability to generalize and trade well in all market conditions. The following table shows ways that overfitting can manifest itself:

Data Practice	Description
Data Dredging	Performing many statistical tests on data and only paying attention to those that come back with significant results.
Hyper-Tuning Parameters	Manually changing algorithm parameters to produce better results without altering the test data.
Overfit Regression Models	Regression, machine learning, or other statistical models with too many variables will likely introduce overfitting to an algorithm.
Stale Testing Data	Not changing the backtesting data set when testing the algorithm. Any improvements might not be able to be generalized to different datasets.

An algorithm that is dynamic and generalizes to new data is more valuable to funds and individual investors. It is more likely to survive across different market conditions and apply to new asset classes and markets.

If you have a collection of factors, you can backtest over a period of time to find the best-performing factors for the time period. If you then narrow the collection of factors to just the best-performing ones and backtest over the same period, the backtest will show great results. However, if you take the same best-performing factors and backtest them on an out-of-sample dataset, the

performance will almost always underperform the in-sample period. To avoid issues with overfitting, follow these guidelines:

- Use walk-forward optimization to train your models on historical data and test them on future data.
- Test your strategy with live paper trading.
- Test your model on different asset classes and markets.

Look-ahead Bias

Look-ahead bias occurs when you use information from the future to inform decisions in the present. An example of look-ahead bias is using financial statement data to make trading decisions at the end of the reporting period instead of when the financial statement data was released. Another example is using updated financial statement data before the updated figures were actually available. Wang et al. (2014) show that using the date of when the period ends instead of when the data is actually available can increase the performance of the earnings yield factor by 60%.

Another culprit of look-ahead bias is adjusted price data. Splits and reverse splits can improve liquidity or attract certain investors, causing the performance of the stock to be different than without the split or reverse split. Wang et al (2014) build a portfolio using the 25 lowest priced stocks in the S&P index based on adjusted and non-adjusted prices. The portfolio based on adjusted prices greatly outperformed the one with raw prices in terms of return and Sharpe ratio . In this case, if you were to analyze the low price factor with adjusted prices, it would lead you to believe the factor is very informative, but it would not perform well out-of-sample with raw data.

Look-ahead bias can also occur if you set the universe to assets that have performed well during the backtest period or initialize indicators with values that have performed well during the backtest period. To avoid issues with look-ahead bias, trade a dynamic universe of assets and use point-in-time data. If point-in-time data is not available for the dataset you use, apply a reporting lag. Since the backtesting environment is event-driven and you can't pass the time frontier, it naturally helps to reduce the risk of look-ahead bias.

Survivorship Bias

Survivorship bias occurs when you test a strategy using only the securities that have survived to the end of the testing period and omit securities that have been delisted. If you use the current constituents of an index and backtest over the past, you'll most likely outperform the index because many of the underperformers have dropped out of the index over time and outperformers have been added to the index. In this case, the current constituent universe would consist of only outperformers. This technique is a form of look-ahead bias because if you were trading the strategy in real-time throughout the backtest period, you would not know the current index constituents until today.

If you analyze a dataset that has survivorship bias, you can discover results that are opposite of the true results. For example, Wang et al. (2014) analyze the low volatility factor using two universes. The first universe was the current S&P 500 constituents and the second universe was the point-in-time constituents. When they used the point-in-time constituents, the low volatility quintile outperformed the high volatility quintile. However, when they used the current constituents, the high volatility quintile significantly outperformed the low volatility quintile.

To avoid issues with survivorship bias, trade a dynamic universe of assets and use the datasets in the Dataset Market . We thoroughly vet the datasets we add to the market to ensure they're free of survivorship bias.

Outliers

Outliers in a dataset can have large impacts on how models train. In some cases, you may leave outliers in a dataset because they can contain useful information. In other cases, you may want to transform the data to handle the outlier data points. There are several common methods to handle outliers.

Winsorization Method

The winsorization method removes outliers at the x% of the extremes. For example, if you winsorize at 1%, it removes the 1% of data points with the lowest value and the 1% of data points with the highest value. This threshold percentage is subjective, so it can result in overfitting.

IQR Method

The interquartile range method removes data points that fall outside of the interval $[Q_1 - k(Q_3 - Q_1),\ Q_3 + k(Q_3 - Q_1)]$ for some constant $k \geq 0$. This method can exclude up to 25% of observations on each side of the extremes. The interquartile range method doesn't work if the data is skewed or non-normal. Therefore, the disadvantage to this method is you need to review the factor distribution properties. If you need to, you can normalize the data with z-scores.

$$Z = \frac{x - \mu}{\sigma}$$

Factor Ranking Method

The factor ranking method ranks the data values instead of taking their raw values. With this method, you don't need to remove any outlier data points. This method transforms the data into a uniform distribution. After converting the data to a uniform distribution, Wang et al. (2014) perform an inverse normal transformation to make the factor values normally distributed. Wang et al. found this ranking technique outperforms the z-score transformation, suggesting that the distance between factor scores doesn't add useful information.

Short Availability

To open a short position, you need to borrow shares from an investor or brokerage that owns them. Wang et al. (2014) show that some factors have a larger edge for long positions and some have a larger edge for short positions. Their research found that the factors that perform better on the short side usually target more securities that are on hard-to-borrow lists than factors that perform better on the long side. Their research also shows that a large part of the returns from these short-biased factors come from securities that are on hard-to-borrow lists. Therefore, if you don't account for short availability, your backtest results can be unrealistic and not reproducible in live trading.

To avoid issues with short availability, add a shortable provider to your algorithm. The shortable provider only lets you open a short trade if there are actually shares available to borrow. By default, LEAN doesn't simulate the cost of borrowing shorts in backtests, but you can enable them by setting a margin interest model.

Examples

The following examples demonstrate some common practices for applying research.

Example 1: Hypothesis-Driven Research

The following example hypothesized that 2 different classes of Google stocks are cointegrated with each other, and we can capitalize spread reversal from occasional, irrational price divergence. To prove it, we make use of the augmented Dickey-Fuller test to verify the cointegration relationship.

C#

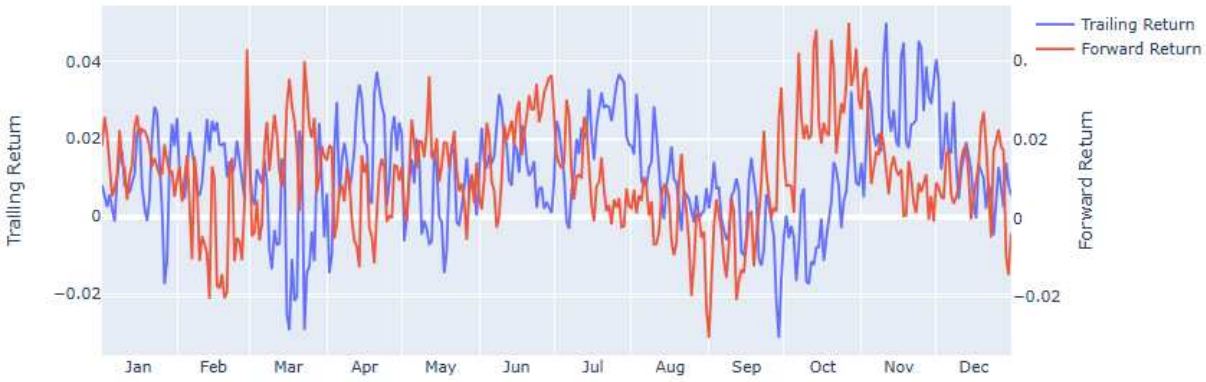
```
using Accord.Statistics;
using MathNet.Numerics.LinearRegression;

public class HypothesisDrivenResearchAlgorithm : QCAlgorithm
{
    private Symbol _goog1, _goog2;
    // The threshold that the spread/residual of the cointegrated series triggers a trade.
    private decimal _thresold;
    // Store the coefficient and intercept of the cointegrated series for calculating the spread of a new data point.
    private decimal[] _coefficients = new[] { 0m, 0m };
    // Store the price series of each symbol for cointegration calculation.
    private Dictionary<Symbol, RollingWindow<double>> _windows = new();
```

```
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Subscribe to 2 classes of Google stocks to trade their price divergence.
        _goog1 = AddEquity("GOOGL", Resolution.Minute).Symbol;      // Class A
        _goog2 = AddEquity("GOOG", Resolution.Minute).Symbol;       // Class C

        foreach (var symbol in new[] { _goog1, _goog2 })
        {
            _windows[symbol] = new(252);

            // Add a consolidator to aggregate a daily bar to update the window's daily price series.
            var consolidator = new TradeBarConsolidator(TimeSpan.FromDays(1));
            consolidator.DataConsolidated += (_, bar) => {
                _windows[bar.Symbol].Add((double)bar.Close);
            };
            // Subscribe the consolidator to update automatically.
            SubscriptionManager.AddConsolidator(symbol, consolidator);

            // Warm up the rolling window's daily price series with historical data.
            var history = History<TradeBar>(symbol, 253, Resolution.Daily);
            foreach (var bar in history)
            {
                consolidator.Update(bar);
            }
        }

        // Adjust the cointegration factor between the 2 classes' monthly price series.
        Schedule.On(
            DateRules.MonthStart(),
            TimeRules.At(0, 1),
            CalculateCointegration
        );

        CalculateCointegration();
    }

    public override void OnData(Slice slice)
    {
        if (slice.QuoteBars.TryGetValue(_goog1, out var bar1) && slice.QuoteBars.TryGetValue(_goog2, out var bar2))
        {
            // Calculate the current cointegrated series spread.
            var residual = _coefficients[0] * bar2.Close + _coefficients[1] - bar1.Close;

            // If the residual is lower than the negative threshold, it means class A price is much higher than what
it should be compared to class C.
            // We sell class A and buy class C to bet on their price convergence.
            if (residual < -_thresold && !Portfolio[_goog1].IsShort)
            {
                SetHoldings(_goog1, -0.5m);
                SetHoldings(_goog2, 0.5m * _coefficients[0]);
            }
            // If the residual is higher than the threshold, it means class A price is much lower than what it should
be compared to class C.
            // We buy class A and sell class C to bet on their price convergence.
            else if (residual > _thresold && !Portfolio[_goog1].IsLong)
            {
                SetHoldings(_goog1, 0.5m);
                SetHoldings(_goog2, -0.5m * _coefficients[0]);
            }
            // Close positions of the price are converged.
            else if ((Portfolio[_goog1].IsShort && residual > 0m) || (Portfolio[_goog1].IsLong && residual < 0m))
            {
                Liquidate();
            }
        }
    }

    private void CalculateCointegration()
    {
        // Lag direction is unimportant; it is just a sign flip in the linear regression, so we don't need to flip
the window order.
        var y = _windows[_goog1].ToArray();
        var x = _windows[_goog2].ToArray();

        // Perform Linear Regression on both price series to investigate their relationship.
        var regressionResult = SimpleRegression.Fit(x, y);
        var intercept = regressionResult.Item1;
```

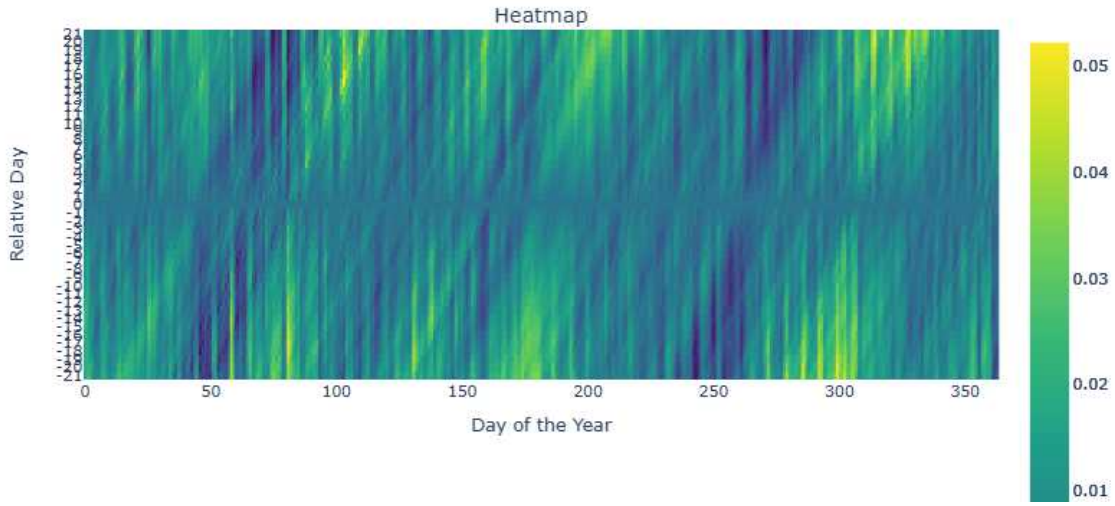
```
        var slope = regressionResult.Item2;

        // Calculate the residuals series to check if it is stationary, meaning if the 2 price series move together.
        var residuals = new double[x.Length];
        for (int i = 0; i < x.Length; i++)
        {
            residuals[i] = y[i] - (intercept + slope * x[i]);
        }

        // Check if the residuals are stationary using the augmented Dickey-Fuller test.
        if (ADFTest(residuals))
        {
            // If cointegrated, update the positional sizing ratio and the spread threshold of the trade trigger.
            _coefficients = new[] { Convert.ToDecimal(slope), Convert.ToDecimal(intercept) };
            _thresold = 2m * Convert.ToDecimal(Measures.StandardDeviation(residuals));
        }
        else
        {
            // If not cointegrated, liquidate and set the size to zeros for no positions.
            Liquidate();
            _coefficients = new[] { 0m, 0m };
            _thresold = 100000000m;             // An arbitrarily large number that the class A price will never
reach.
        }
    }

    private static bool ADFTest(double[] series)
    {
        var n = series.Length;
        var lagged = new double[n - 1];
        var differences = new double[n - 1];

        // Fit linear regression for the residual series on unit root: ΔY_t = α + βY_{t-1} + ε_t.
        for (int i = 1; i < n; i++)
        {
            lagged[i - 1] = series[i - 1];
            differences[i - 1] = series[i] - series[i - 1];
        }

        var regressionResult = SimpleRegression.Fit(lagged, differences);
        var alpha = regressionResult.Item1;  // Intercept
        var beta = regressionResult.Item2;   // Coefficient of lagged term

        // Calculate the ADF statistic and check if the null hypothesis is rejected.
        var adfStatistic = beta / Measures.StandardError(differences);

        // Reject the null hypothesis of a unit root is present if test statistic <= -3.45 (approximate α=0.05 for
n=250)
        // Which means no unit root for difference series and the residuals are stationary.
        return adfStatistic <= -3.45d;
    }
}
```

Example 2: Data-Driven Research

By exploring the return pattern of SPY in 2020, we use that information to invest in 2021. Since the macroeconomic environment is similar (low interest rate), we assume the market seasonality would follow through. We can observe from the figures below that an around 50-day cycle, like a sine function, exists, and the next cycle starts around Jan 2021. Hence, we switch long and short every 25 days in 2021.



Return Analysis



C#

```
public class DataDrivenResearchAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private ScheduledEvent _lastScheduledEvent;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Request SPY data to trade it.
        _spy = AddEquity("SPY", Resolution.Minute).Symbol;
        // Add a warm-up period so SPY has a price before the first trade.
        SetWarmUp(TimeSpan.FromDays(7));
    }

    public override void OnWarmupFinished()
    {
        // According to the data, the first cycle is downgoing.
        SetHoldings(_spy, -1m);
        // Schedule a switch in 25 days later.
        _lastScheduledEvent = Schedule.On(
            DateRules.On(Time.AddDays(25)),
            TimeRules.At(9, 30),
            Switch
        );
    }

    private void Switch()
    {
        // Switch long/short after the cycle change.
        if (Portfolio[_spy].IsLong)
        {
            SetHoldings(_spy, -1m);
        }
        else
        {
            SetHoldings(_spy, 1m);
        }

        // Schedule the next switch in 25 days later.
        Schedule.Remove(_lastScheduledEvent);
        _lastScheduledEvent = Schedule.On(
            DateRules.On(Time.AddDays(25)),
            TimeRules.At(9, 30),
            Switch
        );
    }
}
```

Key Concepts

Globals and Statics

Introduction

It is often convenient to access the QCAlgorithm API in classes outside of the QCAlgorithm class. To achieve this, pass an instance reference or access a global static variable. Whenever possible, we recommend using an instance reference to keep your global namespace clean.

API Access by Algorithm Instance

You can access the QCAlgorithm API by passing the `this` object into a constructor of your target class. The class constructor receives it as a variable you can use to initialize your algorithm. The following algorithm demonstrates this process:

C#

```
public class CustomPartialFillModelAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private SecurityHolding _holdings;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request SPY data for trading.
        var equity = AddEquity("SPY", Resolution.Hour);
        _spy = equity.Symbol;
        _holdings = equity.Holdings;

        // Set the fill model with the algorithm instance.
        equity.SetFillModel(new CustomPartialFillModel(this));
    }

    public override void OnData(Slice data)
    {
        // To get all SPY open orders, use the transaction manager.
        var openOrders = Transactions.GetOpenOrders(_spy);
        if (openOrders.Count != 0) return;

        // Hold SPY during 10th-20th in each month.
        if (Time.Day > 10 && _holdings.Quantity <= 0)
        {
            MarketOrder(_spy, 105, true);
        }
        else if (Time.Day > 20 && _holdings.Quantity >= 0)
        {
            MarketOrder(_spy, -100, true);
        }
    }

    internal class CustomPartialFillModel : FillModel
    {
        private readonly QCAlgorithm _algorithm;
        private readonly Dictionary<int, decimal> _absoluteRemainingByOrderId = new();

        public CustomPartialFillModel(QCAlgorithm algorithm)
            : base()
        {
            _algorithm = algorithm;
        }

        public override OrderEvent MarketFill(Security asset, MarketOrder order)
        {
            if (!_absoluteRemainingByOrderId.TryGetValue(order.Id, out var absoluteRemaining))
            {
                absoluteRemaining = order.AbsoluteQuantity;
            }
```

```
            }

            var fill = base.MarketFill(asset, order);

            // Allow at most 10 shares to be filled at once.
            fill.FillQuantity = Math.Sign(order.Quantity) * 10m;

            // If the remaining shares are less than or equal to 10, the order will be fully filled.
            if (Math.Min(Math.Abs(fill.FillQuantity), absoluteRemaining) == absoluteRemaining)
            {
                fill.FillQuantity = Math.Sign(order.Quantity) * absoluteRemaining;
                fill.Status = OrderStatus.Filled;
                _absoluteRemainingByOrderId.Remove(order.Id);
            }
            // Otherwise, the order is only partially filled out, and we need to update the remaining quantity.
            else
            {
                fill.Status = OrderStatus.PartiallyFilled;
                _absoluteRemainingByOrderId[order.Id] = absoluteRemaining - Math.Abs(fill.FillQuantity);
                var price = fill.FillPrice;

                // Use the instance of the algorithm to log the information
                _algorithm.Debug($"{_algorithm.Time} - Partial Fill - Remaining
{_absoluteRemainingByOrderId[order.Id]} Price - {price}");
            }
            return fill;
        }
    }
}
```

API Access by Global Static Variable

Occasionally, passing the QCAlgorithm object to your class constructor is impossible, so you need to use global static variables to access the API for debugging and initialization purposes. The most common case is custom data implementations , where the LEAN Engine is creating the objects, and you're unable to specify a custom constructor.

To create a global static variable, assign the reference to the global variable in the algorithm's `Initialize` method. This assignment sets the active instance of the variable to the global static. Then, in your custom class, you can access the QCAlgorithm API through the global. The following algorithm demonstrates this process:

C#

```
public class MyAlgorithmInstance : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2012, 9, 1);
        SetEndDate(2024, 12, 31);

        // Instantiate a custom type as a global variable in the QCAlgorithm instance.
        // Assign self to Static Cape Variable.
        Cape.ALGORITHM = this;
        // Request data for trading or signal.
        _symbol = AddData<Cape>("CAPE").Symbol;
    }

    public override void OnData(Slice data)
    {
        if (data.TryGet<Cape>(_symbol, out var cape))
            Plot("CAPE", "Value", cape.Value);
    }
}

public class Cape : BaseData
{
    public static QCAlgorithm ALGORITHM;

    public override SubscriptionDataSource GetSource(SubscriptionDataConfig config, DateTime date, bool isLiveMode)
    {
        ALGORITHM.Debug("Test Static: GetSource");
        return new
SubscriptionDataSource("https://raw.githubusercontent.com/QuantConnect/Documentation/master/Resources/datasets/custom-
data/CAPE.csv", SubscriptionTransportMedium.RemoteFile);
    }

    public override BaseData Reader(SubscriptionDataConfig config, string line, DateTime date, bool isLiveMode)
    {
        var data = line.Split(',').ToArray(x => x.Trim());
        if (data.Length < 12 || line.StartsWith("Date"))
            return null;

        try
        {
            // Construct the custom data type with input data entry.
            return new Cape()
            {
                Symbol = config.Symbol,
                Value = data[12].IfNotNullOrEmpty(
                    s => decimal.Parse(s, NumberStyles.Any, CultureInfo.InvariantCulture)),
                Time = DateTime.ParseExact(data[0], "yyyy.M", CultureInfo.InvariantCulture),
            };
        }
        catch (Exception e)
        {
            ALGORITHM.Debug($"Test Static 2: ValueError: {e.Message}, line: {line}");
        }
        return null;
    }
}
```

Key Concepts

Libraries

Introduction

Libraries (or packages) are third-party software that you can use in your projects. You can use many of the available open-source libraries to complement the classes and methods that you create. Libraries reduce your development time because it's faster to use a pre-built, open-source library than to write the functionality. Libraries can be used in backtesting, research, and live trading. The environments support various libraries for machine learning, plotting, and data processing. As members often request new libraries, we frequently add new libraries to the underlying docker image that runs the Lean engine.

This feature is primarily for Python algorithms as not all Python libraries are compatible with each other. We've bundled together different sets of libraries into distinct environments. To use the libraries of an environment, set the environment in your project and add the relevant `using` statement of a library at the top of your file.

Default Environment Libraries

The default environment supports the following libraries:

```C#
Accord                              3.6.0
Accord.Fuzzy                        3.6.0
Accord.MachineLearning              3.6.0
Accord.Math                         3.6.0
Accord.Statistics                   3.6.0
CloneExtensions                     1.3.0
Common.Logging                      3.4.1
Common.Logging.Core                 3.4.1
CsvHelper                           19.0.0
Deedle                              2.1.0
DotNetZip                           1.16.0
DynamicInterop                      0.9.1
fasterflect                         3.0.0
MathNet.Numerics                    5.0.0
McMaster.Extensions.CommandLineUtils 2.6.0
Microsoft.IO.RecyclableMemoryStream  3.0.1
Microsoft.NET.Test.Sdk              16.9.4
Microsoft.TestPlatform.ObjectModel  16.9.4
Moq                                 4.16.1
NetMQ                               4.0.1.6
Newtonsoft.Json                     13.0.2
NodaTime                            3.0.5
NUnit                               4.2.2
NUnit3TestAdapter                   4.6.0
protobuf-net                        3.1.3
QLNet                               1.13.1
QuantConnect.pythonnet              2.0.48
RestSharp                           106.12.0
SharpZipLib                         1.3.3
System.ComponentModel.Composition   6.0.0
```

Legacy Environment Libraries

This environment is only available for Python projects.

Autogluon Environment Libraries

This environment is only available for Python.

Add New Libraries

To request a new library, contact us . We will add the library to the queue for review and deployment. Since the libraries run on our servers, we need to ensure they are secure and won't cause harm. The process of adding new libraries takes 2-4 weeks to complete. View the list of libraries currently under review on the Issues list of the Lean GitHub repository .

Project Libraries

Project libraries are QuantConnect projects you can merge into your project to avoid duplicating code files. If you have tools that you use across several projects, create a library.

The process to create a library depends on if you use the Cloud Platform , Local Platform , or CLI .

Key Concepts

Debugging Tools

Introduction

When your algorithm throws errors or doesn't operate how you expect, use debugging tools to diagnose and resolve the issue. The tools to debug LEAN algorithms include a built-in debugger, logs, charts, and the Object Store.

Debugger

The debugger is a built-in tool to help you debug coding errors while backtesting. The debugger enables you to slow down the code execution, step through the program line-by-line, and inspect the variables to understand the internal state of the program. The learn how to start and operate the backtest debugger in your development environment, see the Debugging page in the Cloud Platform , Local Platform , or CLI documentation.

Logs

Algorithms can record string messages ('log statements') to a file for analysis after a backtest is complete, or as a live algorithm is running. These records can assist in debugging logical flow errors in the project code or recording key decision moments. Backtests can process millions of data points. If you place a log statement in the wrong place, it can generate gigabytes of logs, which can be difficult to sift through. When you add to the log file, ensure they are readable.

```C#
// Logging records messages to a file so you can debug errors and record key decisions.
Log($"Signal triggered. Placing limit order: {limit}");
```

Debug statements are the same as log statements, but `Debug` method statements are orange in the Cloud Terminal to provide more emphasis. Debug statements are streamed to the Cloud Terminal while the backtest is running. This can be helpful for real-time updates or monitoring in live-trading.

```C#
// Debugging streams messages to the Cloud Terminal for real-time analysis.
Debug($"Exposure to the {sectorName} sector is above 50%!");
```

For more information about log statements, see Logging .

Charts

Plotting values over time can help you understand its range, identify outliers in data, and see times when your algorithm generates unusual values. Before you create a plot, develop a theory about what you expect to see before you view it to gain a deeper understanding of your algorithm. For example, a plot of an asset's price standard deviations should be between -4 and +4, with any outliers happening during extreme market conditions like the COVID pandemic in March 2020. LEAN provides a powerful charting API. You can create time series plots with just a single line of code.

```C#
// Create time series plots to understand the asset range and identify outliers.
Plot("Deviations", "aapl", assetDeviation);
```

For more information about creating charts, see Charting .

Object Store

The Object Store is a key-value data store for low-latency information storage and retrieval. During a backtest, you can build large objects you'd like to analyze and write them for later analysis. This workflow can be helpful when the objects are large and plotting is impossible or when you want to perform analysis across many backtests.

```C#
// The Object Store provides a key-value data store optimized for fast access and retrieval of information.
ObjectStore.Save("key", value);
```

For more information about the Object Store, see Object Store . For a specific example of saving indicator values during a backtest into the Object Store and then plotting them in the Research Environment, see Example for Plotting .

Profiling Speed

QuantConnect provides a series of metrics to help you analyze the computational performance of your algorithms. For more information, see Performance Chart .

JIT Compilation

JIT Compilation is only relevant for Python users.

Key Concepts

Glossary

Introduction

This page defines terms in QuantConnect products and documentation.

alpha

The quantity of an algorithm's returns that aren't explained by its underlying benchmark.

annual standard deviation

A staticial measure that describes the dispersion of annual returns relative to the mean annual return. It's the square root of the annual variance.

annual variance

A staticial measure that describes the dispersion of annual returns relative to the mean annual return.

average loss

The average rate of return for unprofitable trades.

average win

The average rate of return for profitable trades.

beta

The scale and direction of an algorithm's returns relative to movements in the underlying benchmark.

capacity

The maximum amount of money an algorithm can trade before its performance degrades from market impact.

compounding annual return

The annual percentage return that would be required to grow a portfolio from its starting value to its ending value.

day trade

Buying and selling the same asset within one trading day.

drawdown

The largest peak to trough decline in an algorithm's equity curve.

equity

The total portfolio value if all of the holdings were sold at current market rates.

expectancy

The expected return per trade.

holdings

The absolute sum of the items in the portfolio.

information ratio

The amount of excess return from the risk-free rate per unit of systematic risk.

intrinsic value

(Call Option) The price of an asset minus the strike price if the price is above the strike price, otherwise zero.

(Put Option) The strike price minus the price of an asset if the price is below the strike price, otherwise zero.

look-ahead bias

The practice of making decisions using information that would not be available until some time in the future.

loss rate

The proportion of trades that were not profitable.

lowest capacity asset

The asset an algorithm traded that has the lowest capacity.

net profit

(Percent) The rate of return across the entire trading period.

(Dollar-value) The dollar-value return across the entire trading period.

out-of-the-money amount

(Call Option) The strike price minus the price of an asset if the price is below the strike price, otherwise zero.

(Put Option) The price of an asset minus the strike price if the price is above the strike price, otherwise zero.

payoff

The profit or loss that an Option buyer or seller makes from a trade.

pattern day trader

A trader who executes four or more day trades in the US Equity market within five business days.

Probabilistic Sharpe ratio

The probability that the estimated Sharpe ratio of an algorithm is greater than a benchmark (1).

profit-loss ratio

The ratio of the average win rate to the average loss rate.

return

The rate of return across the entire trading period.

risk-free interest rate

The interest rate an investor can expect to earn on an investment that carries zero risk such as the interest paid on a 10-year highly rated government treasury note

Sharpe ratio

A measure of the risk-adjusted return, developed by William Sharpe.

Sortino ratio

A measure of the risk-adjusted return, developed by Frank Sortino. It only penalizes returns falling below a user-specified target or required rate of return.

total fees

The total quantity of fees paid for all the transactions.

total net profit

The rate of return across the entire trading period.

total trades

The number of orders that were filled or partially filled.

tracking error

A measure of how closely a portfolio follows the index to which it is benchmarked. A tracking error of 0 is a perfect match.

Treynor ratio

A measurement of the returns earned in an algorithm in excess of the risk-free rate per unit of benchmark risk, developed by Jack Treynor.

unrealized

The amount of profit a portfolio would capture if it liquidated all open positions and paid the fees for transacting and crossing the spread.

volume

The total value of assets traded for all of an algorithm's transactions.

win rate

The proportion of trades that were profitable after transaction fees.

Initialization

Introduction

The `Initialize` method is the entry point of your algorithm where you define a series of settings, including security subscriptions, starting cash balances, and warm-up periods. LEAN only calls the `Initialize` method one time, at the start of your algorithm.

Set Dates

To set the date range of backtests, call the `SetStartDate` and `SetEndDate` methods. The dates you provide are based in the algorithm time zone. By default, the end date is yesterday, one millisecond before midnight. In live trading, LEAN ignores the start and end dates.

```C#
// Set the start and end dates for the algorithm, defining the backtesting period.
SetStartDate(2013, 1, 5);                 // Set the start date to January 5, 2013.
SetEndDate(2015, 1, 5);                   // Set the end date to January 5, 2015.
SetEndDate(DateTime.Now.Date.AddDays(-7)); // Set the end date to last week.
```

Organization managers can enforce all backtests to end a certain number of months before the current date to reduce the chance of overfitting. For more information, see Out-of-Sample Period .

Set Account Currency

The algorithm equity curve, benchmark, and performance statistics are denominated in the account currency. To set the account currency and your starting cash, call the `SetAccountCurrency` method. By default, the account currency is USD and your starting cash is $100,000. If you call the `SetAccountCurrency` method, you must call it before you call the `SetCash` method or add data . If you call the `SetAccountCurrency` method more than once, only the first call takes effect.

```C#
// Set the account currency and your starting cash.
SetAccountCurrency("BTC"); // Set the account currency to Bitcoin and its quantity to 100,000 BTC.
SetAccountCurrency("INR"); // Set the account currency to Indian Rupees and its quantity to 100,000 INR.
SetAccountCurrency("BTC", 10);  // Set the account currency to Bitcoin and its quantity to 10 BTC.
```

Set Cash

To set your starting cash in backtests, call the `SetCash` method. By default, your starting cash is $100,000 USD. In live trading, LEAN ignores the `SetCash` method and uses the cash balances in your brokerage account instead.

```C#
// Set the initial cash balance for the account for supported currencies.
SetCash(100000);       // Set the quantity of the account currency to 100,000.
SetCash("BTC", 10);    // Set the Bitcoin quantity to 10.
SetCash("EUR", 10000); // Set the EUR quantity to 10,000.
```

Set Brokerage and Cash Model

We model your algorithm with margin modeling by default, but you can select a cash account type. Cash accounts don't allow leveraged trading, whereas Margin accounts can support leverage on your account value. To set your brokerage and account type, call the `SetBrokerageModel` method. For more information about each brokerage and the account types they support, see

the brokerage integration documentation. For more information about the reality models that the brokerage models set, see Supported Models .

```
// Set the brokerage model and account type to model fees, supported order types, and available margin.
SetBrokerageModel(BrokerageName.InteractiveBrokersBrokerage, AccountType.Margin);
```

The `AccountType` enumeration has the following members:

Set Universe Settings

The universe settings of your algorithm configure some properties of the universe constituents. The following table describes the properties of the `UniverseSettings` object:

Property: `Asynchronous` Should the universe selection run asynchronously to boost speed? Data Type: `bool` | Default Value: `false`
Property: `ExtendedMarketHours` Should assets also feed extended market hours? You only receive extended market hours data if you create the subscription with an intraday resolution. If you create the subscription with daily resolution, the daily bars only reflect the regular trading hours. Data Type: `bool` | Default Value: `false`
Property: `FillForward` Should asset data fill forward? Data Type: `bool` | Default Value: `true`
Property: `MinimumTimeInUniverse` What's the minimum time assets should be in the universe? Data Type: `TimeSpan` | Default Value: `TimeSpan.FromDays(1)`
Property: `Resolution` What resolution should assets use? Data Type: `Resolution` | Default Value: `Resolution.Minute`

Method: `Schedule.On` What days should universe selection run? If it's `null` , it's the data resolution. For example, US Equity fundamentals universes are daily at midnight. Argument Type: `IDateRule` | Default Value: `null`
Property: `ContractDepthOffset` What offset from the current front month should be used for continuous Future contracts ? 0 uses the front month and 1 uses the back month contract. This setting is only available for Future assets. Data Type: `int` | Default Value: `0`
Property: `DataMappingMode` How should continuous Future contracts be mapped? This setting is only available for Future assets. Data Type: `DataMappingMode` | Default Value: `DataMappingMode.OpenInterest`
Property: `DataNormalizationMode data_normalization_mode` How should historical prices be adjusted? This setting is only available for Equity and Futures assets. Data Type: `DataNormalizationMode` | Default Value: `DataNormalizationMode.Adjusted`
Property: `Leverage` What leverage should assets use in the universe? This setting is not available for derivative assets. Data Type: `decimal` | Default Value: `Security.NullLeverage`

To set the `UniverseSettings` , update the preceding properties in the `Initialize` method before you add the universe. These settings are globals, so they apply to all universes you create.

```
C#
// Request second resolution data. This will be slow!
UniverseSettings.Resolution = Resolution.Second;
AddUniverse(Universe.Top(50));
```

For more information about universe settings, see the related documentation for classic and framework algorithms.

Set Security Initializer

Instead of configuring global universe settings, you can individually configure the settings of each security in the universe with a security initializer. Security initializers let you apply any security-level reality model or special data requests on a per-security basis. To set the security initializer, in the `Initialize` method, call the `SetSecurityInitializer` method and then define the security initializer.

```
// A custom security initializer can override default models such as
// setting new fee and fill models for the security.
SetSecurityInitializer(CustomSecurityInitializer);

private void CustomSecurityInitializer(Security security)
{
    security.SetFeeModel(new ConstantFeeModel(0, "USD"));
}
```

For simple requests, you can use the functional implementation of the security initializer. This style lets you configure the security object with one line of code.

```
// Disable the trading fees for each security by passing a functional
// implementation for the SetSecurityInitializer argument.
SetSecurityInitializer(security => security.SetFeeModel(new ConstantFeeModel(0, "USD")));
```

If you call the `SetSecurityInitializer` method, it overwrites the default security initializer. The default security initializer uses the security-level reality models of the brokerage model to set the following reality models of each security:

- Fill
- Slippage
- Fee
- Buying Power
- Settlement
- Short availability
- Margin Interest Rate

The default security initializer also sets the leverage of each security and intializes each security with a seeder function. To extend upon the default security initializer instead of overwriting it, call `AddSecurityInitializer` before you create the subscriptions.

```
public class AddSecurityInitializerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the security initializer to set models of assets.
        AddSecurityInitializer(CustomSecurityInitializer);
    }

    private void CustomSecurityInitializer(Security security)
    {
        // Overwrite some of the reality models
        security.SetFeeModel(new ConstantFeeModel(0, "USD"));
    }
}
```

Add Data

You can subscribe to asset, fundamental, alternative, and custom data. The Dataset Market provides 400TB of data that you can easily import into your algorithms.

Asset Data

To subscribe to asset data, call one of the asset subscription methods like `AddEquity` or `AddForex` . Each asset class has its own method to create subscriptions. For more information about how to create subscriptions for each asset class, see Asset Classes

.

```
AddEquity("AAPL"); // Add Apple 1 minute bars (minute by default)
AddForex("EURUSD", Resolution.Second); // Add EURUSD 1 second bars
```

In live trading, you define the securities you want, but LEAN also gets the securities in your live portfolio and sets their resolution to the lowest resolution of the subscriptions you made. For example, if you create subscriptions in your algorithm for securities with Second, Minute, and Hour resolutions, the assets in your live portfolio are given a resolution of Second.

Alternative Data

To add alternative datasets to your algorithms, call the `AddData` method. For full examples, in the Datasets chapter, select a dataset and see the Requesting Data section.

Custom Data

To add custom data to your algorithms, call the `AddData` method. For more information about custom data, see Importing Data .

Limitations

There is no official limit to how much data you can add to your algorithms, but there are practical resource limitations. Each security subscription requires about 5MB of RAM, so larger machines let you run algorithms with bigger universes. For more information about our cloud nodes, see Resources .

Set Indicators and Consolidators

You can create and warm-up indicators in the `Initialize` method.

```
private Symbol _symbol;
private SimpleMovingAverage _sma;

_symbol = AddEquity("SPY").Symbol;
_sma = SMA(_symbol, 20);
WarmUpIndicator(_symbol, _sma);
```

Set Algorithm Settings

The following table describes the `AlgorithmSettings` properties:

Property: `AutomaticIndicatorWarmUp` A flag that defines if the `WarmUpIndicator` method can warm up indicators. Data Type: `bool` | Default Value: `false`
Property: `SeedInitialPrices` Determines whether to seed initial prices for all selected and manually added securities. Data Type: `bool` | Default Value: `false`

Property: `DailyPreciseEndTime`

A flag that defines if daily bars should have an EndTime that matches the market close time (`true`) or the following midnight (`false`).

Data Type: `bool` | Default Value: `true`

Property: `FreePortfolioValue`

The buying power buffer value.

Data Type: `decimal` | Default Value: `250 m`

Property: `FreePortfolioValuePercentage`

The buying power buffer percentage value.

Data Type: `decimal` | Default Value: `0.0025 m`

Property: `IgnoreUnknownAssetHoldings`

Determines whether to terminate the algorithm when an asset holding is not supported by Lean or the brokerage.

Data Type: `bool` | Default Value: `true`

Property: `LiquidateEnabled`

A flag to enable and disable the `Liquidate` method.

Data Type: `bool` | Default Value: `true`

Property: `MaxAbsolutePortfolioTargetPercentage`

The absolute maximum valid total portfolio value target percentage.

Data Type: `decimal` | Default Value: `1000000000 m`

Property: `MinAbsolutePortfolioTargetPercentage`

The absolute minimum valid total portfolio value target percentage.

Data Type: `decimal` | Default Value: `0.0000000001 m`

Property: `MinimumOrderMarginPortfolioPercentage`

The minimum order margin portfolio percentage to ignore bad orders and orders with small sizes.

Data Type: `decimal` | Default Value: `0.001 m`

Property: `PerformanceSamplePeriod`

Performance tracking sample period to use if any, useful to debug performance issues. See Performance Chart .

Data Type: `TimeSpan` | Default Value: `TimeSpan.Zero`

Property: `RebalancePortfolioOnInsightChanges`

Rebalance the portfolio when you emit new insights or when insights expire.

Data Type: `bool/NoneType` | Default Value: `true`

Property: `RebalancePortfolioOnSecurityChanges`

Rebalance the portfolio when your universe changes.

Data Type: `bool/NoneType` | Default Value: `true`

Property: `StalePriceTimeSpan`

The minimum time span elapsed to consider a market fill price as stale

Data Type: `TimeSpan` | Default Value: `TimeSpan.FromHours(1)`

Property: `TradingDaysPerYear`

Number of trading days per year for this algorithm's portfolio statistics .

Data Type: `integer` | Default Value: `252`

Property: `WarmUpResolution`

The resolution to use during the warm-up period

Data Type: `Resolution/NoneType` | Default Value: `null`

To change the `Settings` , update some of the preceding properties.

```
// Set the algorithm behavior using the Settings property on your algorithm.
Settings.RebalancePortfolioOnSecurityChanges = false;
Settings.TradingDaysPerYear = 365;
```

To successfully update the `FreePortfolioValue` , you must update it after the `Initialize` method.

Set Benchmark

The benchmark performance is input to calculate several statistics on your algorithm, including alpha and beta . To set a benchmark for your algorithm, call the `SetBenchmark` method. You can set the benchmark to a security, a constant value, or a value from a custom data source . If you don't set a brokerage model , the default benchmark is SPY. If you set a brokerage model, the model defines the default benchmark.

```
// Set the benchmark to IBM
SetBenchmark("IBM");

// Set the benchmark to a constant value of 0
SetBenchmark(x => 0);

// Set the benchmark to a value from a custom data source
var symbol = AddData<CustomData>("CustomData", Resolution.Hour).Symbol;
SetBenchmark(symbol);
```

If you pass a ticker to the `SetBenchmark` method, LEAN checks if you have a subscription for it. If you have a subscription for it, LEAN uses the security subscription. If you don't have a subscription for it, LEAN creates a US Equity subscription with the ticker. Since the ticker you pass may not reference a US Equity, we recommend you subscribe to the benchmark security before you call the `SetBenchmark` method.

Set Time Zone

LEAN supports international trading across multiple time zones and markets, so it needs a reference time zone for the algorithm to set the `Time` . The default time zone is Eastern Time (ET), which is UTC-4 in summer and UTC-5 in winter. To set a different time zone, call the `SetTimeZone` method. This method accepts either a string following the IANA Time Zone database convention or a NodaTime .DateTimeZone object. If you pass a string, the method converts it to a `NodaTime.DateTimeZone` object. The `TimeZones` class provides the following helper attributes to create `NodaTime.DateTimeZone` objects:

```
SetTimeZone("Europe/London");
SetTimeZone(NodaTime.DateTimeZone.Utc);
SetTimeZone(TimeZones.Chicago);
```

The algorithm time zone may be different from the data time zone . If the time zones are different, it might appear like there is a lag between the algorithm time and the first bar of a history request, but this is just the difference in time zone. All the data is internally synchronized in Coordinated Universal Time (UTC) and arrives in the same Slice object. A slice is a sliver of time with all the data available for this moment.

To keep trades easy to compare between asset classes, we mark all orders in UTC time.

Set Warm Up Period

You may need some historical data at the start of your algorithm to prime technical indicators or populate historical data arrays. The warm-up period pumps data into your algorithm from before the start date. To set a warm-up period, call the `SetWarmUp` method. The warm-up feature uses the subscriptions you add in `Initialize` to gather the historical data that warms up the

algorithm. If you don't create security subscriptions in the `Initialize` method, the warm-up won't occur.

```
C#

// Wind time back 7 days from the start date
SetWarmUp(TimeSpan.FromDays(7));

// Feed in 100 trading days worth of data before the start date
SetWarmUp(100, Resolution.Daily);

// If you don't provide a resolution argument, it uses the lowest resolution in your subscriptions
SetWarmUp(100);
```

Set Name and Tags

You can categorize your backtest with a name and tags. To set the algorithm `Name` , call the `SetName` method.

```
C#

// Set the name of your backtest to help categorize your results.
SetName("Backtest Name");
```

The `SetName` method overwrites the names of backtests of an optimization , which contains the parameter values. To keep these values, create a name that includes them.

```
C#

// Setting the backtest name to a mix of the parameters can help identify backtests in optimization results.
var fastPeriod = GetParameter("fast_period", 100);
var midPeriod = GetParameter("mid_period", 200);
var slowPeriod = GetParameter("slow_period", 300);
SetName($"Backtest Name ({fastPeriod},{midPeriod},{slowPeriod})");
```

To add tags to you algorithm, call the `AddTag` method.

```
C#

// Add additional information to the algorithm with tags for better organization in the backtest results.
AddTag("Long Only");
AddTag("Momentum");
```

Your algorithm can add 20 tags. The name and each tag can have up to 200 characters or else they are truncated.

Post Initialization

After the `Initialize` method, the `PostInitialize` method performs post-initialization routines, so don't override it. To be notified when the algorithm is ready to begin trading, define an `OnWarmupFinished` method. This method executes even if you don't set a warm-up period.

```
C#

// Notify yourself when the algorithm is ready to begin trading.
public override void OnWarmupFinished()
{
    Log("Algorithm Ready");
}
```

Examples

The following examples demonstrate some common practices for initializing algorithms.

Example 1: Set the Data Seed

Some data feeds might not have data on its first fetch, making the algorithm prone to error. The following algorithm seeds the initial price of each asset to its last known price to avoid issues.

```
C#
// Seed with the last known price of assets to prevent $0 security price errors.
public override void Initialize()
{
    Settings.SeedInitialPrices = true;
}
```

Example 2: Enable Extended Market Hours

Some traders use pre-market data for filtering assets. The following algorithm enables extended market hour data for assets in the universe.

```
C#
// Add a universe of assets with extended market hours data.
public override void Initialize()
{
    UniverseSettings.ExtendedMarketHours = true;
}
```

Example 3: Train Models When the Warm Up Finishes

The following algorithm uses the `OnWarmupFinished` event handler to train a model with data from the warm-up period.

```
C#
// Train machine learning models after the algorithm warm up is done, so your models
// are ready as soon as the algorithm starts trading.
public override void OnWarmupFinished()
{
    Train(MyTrainingMethod);
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
BasicTemplateAlgorithm.cs C# BasicTemplateCryptoAlgorithm.cs C# BasicTemplateForexAlgorithm.cs C#
BasicTemplateFuturesAlgorithm.cs C# BasicTemplateOptionsAlgorithm.cs C#

Securities

Securities

Key Concepts

Introduction

A security is an individual financial asset that you can trade on an exchange. LEAN models these unique assets with a `Security` object, which the `AddEquity` , `AddCrypto` , and similar methods return. The `Securities` property of the `QCAlgorithm` class is a dictionary where the keys are `Symbol` objects and the values are `Security` objects. The `Securities` dictionary contains all of the securities that have been in the algorithm during its lifetime.

```
C#
// Add the SPY.
private Symbol _symbol;
_symbol = AddEquity("SPY").symbol;
```

Price

To get the current price of a security, index the `Securities` dictionary and then access the `Price` member.

```
C#
// Get the price of the security.
var price = Securities[_symbol].Price;
```

Holdings

The `holdings` property of a `Security` object contains information about the investment state and history of the security.

```
C#
// Use the holdings object to get information on the asset position, such as the quantity and average price.
var holdings = Securities[_symbol].Holdings;
var quantity = holdings.Quantity;
```

Alternatively, you can get the current holdings of a security with the `Portfolio` dictionary.

```
C#
// Get the quantity of a security you hold through the Portfolio dictionary.
var holdings = Portfolio[_symbol];
var quantity = holdings.Quantity;
```

Reality Models

`Security` objects contain references to the security level reality models . To customize the behavior of each security, configure its reality models.

Tradable Status

The `IsTradable` property shows whether you can trade a security. The property value is true when the security is in the universe, even if the data starts at a later day. Indices, canonical Option securities, and continuous Futures contracts are not tradable. In live mode, custom data objects are also not tradable, even if the custom data represents a tradable asset.

```
C#

// Check if the security is tradable.
var tradable = Securities[_symbol].IsTradable;
```

Symbol Properties

The `SymbolProperties` property of a `Security` contains properties for that specific security, such as the lot size and the minimum price variation.

```
C#

// Round orders to a brokerage's lot size and minimum price variation using the symbol_properties.
var symbolProperties = Securities[_symbol].SymbolProperties;
var lotSize = symbolProperties.LotSize;
var minimumPriceVariation = symbolProperties.MinimumPriceVariation;
```

Market Hours

The `Exchange` property of a `Security` contains information about the exchange that lists the security. For example, the `Exchange.Hours` property contains information on the trading days and hours .

```
C#

// Using the exchange object, check if the exchange's regular trading hours are open at the given time.
var exchange = Securities[_symbol].Exchange;
var hours = exchange.Hours;
var isOpenNow = hours.isOpen(Time, extendedMarketHours: false);
var minimumPriceVariation = symbolProperties.MinimumPriceVariation;
```

Custom Security Properties

You can add properties to the `Security` object. For example, you can add an exponential moving average .

```
C#

// Cast the Equity to a dynamic object to add an EMA indicator to it.
dynamic equity = AddEquity("SPY");
equity.ema = EMA(equity.Symbol, 10, Resolution.Daily);
```

This feature is helpful because you can get the `Security` object from the `Securities` object.

```
C#

// Get the SPY Equity object from Securities to get the EMA indicator.
var ema = (decimal)(Securities["SPY"] as dynamic).ema.Current.Value;
```

To avoid casting to and from the `dynamic` type, you can use the `Set` method to assign and the `Get<T>` method to access the dynamic member and return the object with its type.

```
C#

var equity = AddEquity("SPY");
// Assign an EMA indicator to the "ema" property
equity.Set("ema", EMA(equity.Symbol, 10, Resolution.Daily));
// Get the SPY Equity object from Securities to get the EMA indicator and its current value
var ema = Securities["SPY"].Get<ExponentialMovingAverage>("ema");   // The type of ema is ExponentialMovingAverage
var emaValue = ema.Current.Value;    // The type of emaValue is decimal
```

If the type or the member name is incorrect, the `Get<T>` method causes a runtime error.

Properties

The `Security` type has the following properties:

Examples

The following examples demonstrate some common practices for securities.

Example 1: Accessing Properties

The following example shows how to access various propeties of the `Security` object.

```C#
public class SecuritiesUsageAlgorithm : QCAlgorithm
{
    private Equity _spy;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        _spy = AddEquity("SPY");
    }

    public override void OnData(Slice slice)
    {
        // OHLCV data.
        var price = _spy.Close;
        var volume = _spy.Volume;
        // Quote data.
        var quoteSize = _spy.BidSize;
        var quotePrice = _spy.AskPrice;
        // Symbol Properties.
        var lotSize = _spy.SymbolProperties.LotSize;
        // Fundamentals cached data.
        var peRatio = _spy.Fundamentals.ValuationRatios.PERatio;
        // Properties, e.g. is tradeble, is market open, leverage, ...
        var marketOpen = _spy.IsMarketOpen;
        var leverage = _spy.Leverage;
        // Models, e.g. margin model, buying power model, ...
        var marginModel = _spy.MarginModel;
        // Portfolio Holdings reference.
        var holding = _spy.Holdings;
    }
}
```

Securities

Requesting Data

Introduction

When you add a security to your algorithm, your algorithm creates a data subscription and receives data updates for that security or custom data source .

Add Assets

To create an Equity subscription, call the `AddEquity` method and pass the `ticker` argument. The `ticker` argument represents the current ticker.

```
C#
// Add the SPY ETF.
var equity = AddEquity("SPY");
```

The `AddEquity` method creates a subscription for a single Equity asset and adds it to your user-defined universe.

For more information about adding assets, see Asset Classes .

Add Universes

Universe selection is the process of selecting a basket of assets you may trade. When you add a universe to your algorithm, LEAN sends a large dataset into a filter function you define. LEAN automatically subscribes to these new assets and adds them to your algorithm.

```
C#
// Use the FundamentalFilterFunction to add the 100 most liquid stocks to the universe.
public class MyFundamentalUniverseAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        UniverseSettings.Asynchronous = true;
        AddUniverse(FundamentalFilterFunction);
    }

    // Get the top 100 stocks by dollar volume.
    private IEnumerable<Symbol> FundamentalFilterFunction(IEnumerable<Fundamental> fundamental)
    {
        return (from c in fundamental
            orderby c.DollarVolume descending
            select c.Symbol).Take(100);
    }
}
```

For more information about universes, see Universes .

Resolutions

Resolution is the duration of time that's used to sample a data source. Market data is sampled and saved in disk with five resolutions. The `Resolution` enumeration has the following members:

- `Resolution.Tick` `Resolution.TICK`
- `Resolution.Second` `Resolution.SECOND`

- `Resolution.Minute` `Resolution.MINUTE`
- `Resolution.Hour` `Resolution.HOUR`
- `Resolution.Daily` `Resolution.DAILY`

For more information about the resolutions of each asset class, follow these steps:

1. Open the Asset Classes documentation.
2. Click an asset class.
3. Click Requesting Data .
4. Scroll down to the Resolutions section.

You can subscribe to daily resolution for an asset for updating indicator and subscribe to another asset for denser resolution for trading using the indicator.

```C#
public override void Initialize()
{
    AddEquity("VIX", Resolution.Daily);
    AddEquity("SPY", Resolution.Minute);
}
```

To get data with multiple or customizable resolutions for an asset, use consolidators .

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

For more information about how to enable and disable fill forward for each asset class, follow these steps:

1. Open the Asset Classes documentation.
2. Click an asset class.
3. Click Requesting Data .
4. Scroll down to the Fill Forward section.

Example: Disable Fill Forward Data

For some illiquid securities or alternative data, using forward filled data might not be an accurate estimation nor usable information. To avoid this, disable fill forward.

```C#
public override void Initialize()
{
    AddData<BrainSentiment7Days>(symbol, fillForward: false);
}
```

Security Changed Events

When you add and remove assets, LEAN notifies your algorithm through the `OnSecuritiesChanged` event handler. The event handler receives a `SecurityChanges` object, which contains references to the added and removed securities. To access the added securities, iterate the `changes.AddedSecurities` property. To get the removed securities, iterate the `changes.RemovedSecurities` property.

```C#
// Use OnSecuritiesChanged to know when an asset is added or removed.
public override void OnSecuritiesChanged(SecurityChanges changes)
{
    // Iterate through the added assets.
    foreach (var security in changes.AddedSecurities)
    {
        Debug($"{Time}: Added {security.Symbol}");
    }

    // Iterate through the removed assets.
    foreach (var security in changes.RemovedSecurities)
    {
        Debug($"{Time}: Removed {security.Symbol}");

        if (security.Invested)
        {
            Liquidate(security.Symbol, "Removed from Universe");
        }
    }
}
```

The preceding example liquidates securities that leave the universe. You can use this event to create indicators when a new security enters the universe.

Remove Subscriptions

To remove a security subscription, call the `RemoveSecurity` method.

```C#
// Remove the security subscription.
RemoveSecurity(_symbol);
```

The `RemoveSecurity` method cancels your open orders for the security and liquidates your holdings.

Examples

The following examples demonstrate some common practices on the subject of requesting data.

Example 1: Add Linked Alternative Data for Universe Constituents

Many traders select universe constituents and then allocate positions with two independent sets of logic. Position sizing usually requires additional information, like alternative data factors. To manage alternative data subscriptions for all the universe constituents, subscribe and unsubscribe from the data in the `OnSecuritiesChanged` method. The following example demonstrates adds the KavoutCompositeFactorBundle when assets enter the universe.

```C#
public class RequestSecuritiesDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Add a universe of the 25 most liquid US Equities.
        AddUniverse(Universe.Top(25));
    }

    public override void OnData(Slice slice)
    {
        // Invest by the updated Kavout Factor data.
        var factorData = slice.Get<KavoutCompositeFactorBundle>();
        if (factorData.Count > 0)
        {
            // Obtain the top 10 factor sum score symbols.
            // Invest by equal weighting to dissipate capital risks.
            var topFactorTargets = factorData.OrderByDescending(x =>
                    x.Value.Growth + x.Value.ValueFactor + x.Value.Quality + x.Value.Momentum +
x.Value.LowVolatility
                )
                .Where(x => Securities[x.Key.Underlying].Price != 0)
                .Take(10)
                .Select(x => new PortfolioTarget(x.Key.Underlying, 0.1m))
                .ToList();
            SetHoldings(topFactorTargets, true);
        }
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        // Iterate securities that entered the universe.
        foreach (dynamic security in changes.AddedSecurities)
        {
            // Subscribe to the factor data for the new security.
            // Save the alternative data symbol in the security object.
            security.AltDataSymbol = AddData<KavoutCompositeFactorBundle>(security.Symbol, Resolution.Daily).Symbol;
        }

        // Iterate securities that left the universe.
        foreach (dynamic security in changes.RemovedSecurities)
        {
            // Unsubscribe from the altnerative data updates.
            RemoveSecurity(security.AltDataSymbol);
        }
    }
}
```

Example 2: Add Linked Custom Data

QuantConnect Cloud has a lot of data, but not everything. To import some external data into your algorithm, define a custom data source . The following algorithm loads Bitstamp's BTCUSD trading activity from a CSV file and plots it:

```C#
// Define the algorithm.
public class LinkedCustomDataExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add Bitcoin and the custom dataset.
        dynamic btc = AddCrypto("BTCUSD");
        btc.dataSymbol = AddData<Bitstamp>(btc.Symbol.Value, Resolution.Daily).Symbol;
        // Plot the price from the custom dataset.
        PlotIndicator("BTC Price", Identity(btc.dataSymbol));
    }
}

// Define the custom data source.
public class Bitstamp : BaseData
{
```

```
{
    // Declare the properties of the custom dataset.
    public int Timestamp = 0;
    public decimal Open = 0;
    public decimal High = 0;
    public decimal Low = 0;
    public decimal Close = 0;
    public decimal Bid = 0;
    public decimal Ask = 0;
    public decimal WeightedPrice = 0;
    public decimal VolumeBTC = 0;
    public decimal VolumeUSD = 0;

    // Define the GetSource method to load the data from its location.
    public override SubscriptionDataSource GetSource(SubscriptionDataConfig config, DateTime date, bool isLiveMode)
    {
        var source = "https://raw.githubusercontent.com/QuantConnect/Documentation/master/Resources/datasets/custom-
data/bitstampusd.csv";
        // RemoteFile in the next line means the data file in online storage.
        return new SubscriptionDataSource(source, SubscriptionTransportMedium.RemoteFile);
    }

    // Define the Reader method to parse the CSV rows.
    public override BaseData Reader(SubscriptionDataConfig config, string line, DateTime date, bool isLiveMode)
    {
        // Ignore empty rows.
        if (string.IsNullOrWhiteSpace(line.Trim()))
        {
            return null;
        }

        // Instantiate a new data object.
        var coin = new Bitstamp() {Symbol = config.Symbol};

        // Ignore non-numeric lines.
        if (!char.IsDigit(line[0]))
        {
            return null;
        }

        // Split the CSV data by its commas.
        var data = line.Split(',');
        coin.Value = data[4].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any, CultureInfo.InvariantCulture));
        // Ignore invalid data.
        if (coin.Value == 0)
        {
            return null;
        }

        // Parse the row.
        coin.Time = DateTime.Parse(data[0], CultureInfo.InvariantCulture);
        coin.EndTime = coin.Time.AddDays(1);
        coin.Open = data[1].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any, CultureInfo.InvariantCulture));
        coin.High = data[2].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any, CultureInfo.InvariantCulture));
        coin.Low = data[3].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any, CultureInfo.InvariantCulture));
        coin.VolumeBTC = data[5].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
        coin.VolumeUSD = data[6].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
        coin.WeightedPrice = data[7].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
        coin.Close = coin.Value;
        return coin;
    }
}
```

Securities

Handling Data

Introduction

If you create a data subscription, your algorithm receives data updates for that security or custom data source .

Get Prices

LEAN uses data updates to set the current prices of a security and compute the contribution of securities you hold to the portfolio value. To get the current prices of a security, find the `Security` object in the `Securities` dictionary and access the price attributes.

```
C#
// Use the Securities object as a dictionary to get the current ask and bid price of the SPY.
var security = Securities["SPY"];
var price = security.Price;
var bidPrice = security.BidPrice;
var askPrice = security.AskPrice;
```

Data Events

The `Slice` that LEAN passes to the `OnData` method represents all of the data for your subscriptions at a single point in time. The `Slice` object contains data like `Tick` objects, `TradeBar` objects, `QuoteBar` objects, corporate actions , and chains for Option and Future contracts. You can use the data in the `Slice` to make trading decisions.

To access data in the `Slice` , index it with the security `Symbol` . If you use the security ticker instead of the `Symbol` , LEAN automatically finds the `Symbol` .

```
C#
// Use the OnData method to get the data for a symbol in the current time slice.
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_symbol))
    {
        var myData = slice[_symbol];
    }
}
```

The following table shows the `Slice` property to index based on the data format you want:

Data Format	Slice Property
`TradeBar`	`Bars`
`QuoteBar`	`QuoteBars`
`Tick`	`Ticks`

```C#
// Get the tradebar from the slice.
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_symbol))
    {
        var bar = slice.Bars[_symbol];
    }
}
```

If you just index the `Slice` instead of the preceding properties, it returns the correct object. For instance, if your data subscription provides `QuoteBar` objects and you index the `Slice` with the security `Symbol` , it returns the `QuoteBar` .

For more information about the `Slice` class, see Timeslices .

Cache

To get the most recent data for a security, call the `GetLastData` method on the `Security` object.

```C#
// Get the most recent data for a security.
var data = Securities["SPY"].GetLastData();
```

The `Cache` property of a `Security` object provides additional price information since you can access `TradeBar` , `QuoteBar` , and `Tick` data.

```C#
// Get additional price information for the security.
var cache = Securities["SPY"].Cache;
var tradeBar = cache.GetData<TradeBar>();
var quoteBar = cache.GetData<QuoteBar>();
var ticks = cache.GetAll<Tick>();
```

Market Session

The `session` property of a `Security` object contains the open, high, low, close, volume, and open interest of trading sessions. The `Session` object is a `RollingWindow` object with size of zero by default. To enable market session tracking, set the size of the `Session` object to 2, so the previous session data is at index 1.

```C#
var session = Securities[_symbol].Session;
session.Size = 2;
var todayOpen = session.Open;
var todayHigh = session.High;
var todayLow = session.Low;
var todayCumulativeVolume = session.Volume;
if (session.IsReady)
{
    var yesterdayClose = session[1].Close;
}
```

Increase the size of the `Session` object to store more session data. You can use cached data in the `session` property instead of requesting historical daily data .

```
var equity = AddEquity("SPY");
// Increase the session lookback to 252 from default (0)
equity.Session.Size = 252;
// Get all closing prices in the session history
var closes = session.OrderBy(x => x.EndTime).Select(x => x.Close);
```

For more information about rolling windows, see Rolling Window .

Examples

The following examples demonstrate some common practices for handling data.

Example 1: Get QuoteBar Data

The following algorithm updates an indicator with the mid-price of a quote bar to increase the accuracy of the indicator values for illiquid assets. Then, make use of the SMA indicator to trade the SMA cross strategy.

```
public class HandlingSecuritiesDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    // Initialize a new instance of a SimpleMovingAverage object.
    private SimpleMovingAverage _indicator = new(250);

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        _symbol = AddEquity("SPY").Symbol;

        SetWarmUp(TimeSpan.FromDays(350));
    }

    public override void OnData(Slice slice)
    {
        // Check if the QuoteBars contain SPY quote data.
        if (!slice.QuoteBars.ContainsKey(_symbol))
        {
            return;
        }
        var quoteBar = slice.QuoteBars[_symbol];
        // Calculate the mid price by averaging the bid and ask price.
        var midPrice = (quoteBar.Bid.Close + quoteBar.Ask.Close) * 0.5m;
        // Update the SMA indicator with the mid price.
        _indicator.Update(quoteBar.EndTime, midPrice);

        // Wait for the algorithm and indicator to be warmed up.
        if (IsWarmingUp || !_indicator.IsReady)
        {
            return;
        }
        // Trade SMA cross strategy.
        SetHoldings(_symbol, _indicator > midPrice ? -0.5m : 0.5m);
    }
}
```

Example 2: Get Tick Data

The following algorithm calculates the bid and ask sizes of the latest SPY quote ticks and then trades at the top of each hour. During each rebalance, it buys SPY if bid size is greater than ask size since supply is greater than demand in this case. If bid size is smaller than ask size, it sells SPY since demand is greater than supply in this case.

C#

```
public class BidAskSizeTickAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    // Set up variables to save the bid and ask sizes.
    private decimal _bidSize = 0m, _askSize = 0m;
    private int _hour = -1;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 10);
        // Seed the price of each asset with its last known price to
        // avoid trading errors.
        Settings.SeedInitialPrices = true;
        _symbol = AddEquity("SPY", Resolution.Tick).Symbol;
    }

    public override void OnData(Slice slice)
    {
        // Check if the Ticks object contain SPY tick data.
        if (slice.Ticks.TryGetValue(_symbol, out var ticks))
        {
            // Iterate all the ticks to calculate the bid and ask sizes.
            foreach (var tick in ticks.Where(tick => tick.TickType == TickType.Quote))
            {
                // Update the bid or ask size.
                _bidSize += tick.BidSize;
                _askSize += tick.AskSize;
            }
        }
        // Trade at the top of each hour.
        if (slice.Time.Hour != _hour)
        {
            // Invest based on supply-demand relationship from all ticks.
            // If bid size is above ask size, supply is greater than
            // demand, which drives the price up. Otherwise, supply is
            // lower than demand, which drives the price down.
            SetHoldings(_symbol, _bidSize > _askSize ? 0.5m : -0.5m);
            // Reset the bid and ask size variables.
            _bidSize = 0;
            _askSize = 0;
            _hour = slice.Time.Hour;
        }
    }
}
```

Example 3: Get US Equity Fundamental Data

The following algorithm gets the latest fundamental data of all the US Equities in the algorithm. We invest in the stocks with the highest PE Ratio equally to ride on the popularity.

```C#
public class HandlingSecuritiesDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request data for a selected list of equities for trading.
        foreach (var ticker in new[] { "AAPL", "MSFT", "TSLA", "GOOG", "AMZN" })
        {
            AddEquity(ticker);
        }
    }

    public override void OnData(Slice slice)
    {
        var peRatios = new Dictionary<Symbol, double>();

        // Iterate active security objects.
        foreach (var security in ActiveSecurities.Values)
        {
            // Get the Fundamental cache.
            var fundamental = security.Fundamentals;
            // Get the sector code.
            var sectorCode = fundamental.AssetClassification.MorningstarSectorCode;

            peRatios[security.Symbol] = fundamental.ValuationRatios.PERatio;
        }

        // Sort by PE ratio to get the most popular stocks.
        var sortedByPeRatio = peRatios.OrderByDescending(x => x.Value)
            .ToDictionary(x => x.Key, x => x.Value);
        var targets = sortedByPeRatio
            .Take(3)
            .Select(x => new PortfolioTarget(x.Key, 1m / 3m))
            .ToList();
        targets.AddRange(sortedByPeRatio
            .TakeLast(2)
            .Select(x => new PortfolioTarget(x.Key, 0m))
            .ToList());
        // Invest equally in the highest PE Ratio stocks for evenly dissipate capital risk.
        SetHoldings(targets);
    }
}
```

Example 4: Get Option Greeks

The following algorithm gets SPY Option Greek values. Sell a straddle with the ATM options, selected by Option Delta (closest to 0.5).

```C#
public class HandlingSecuritiesDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Add an SPY Option universe.
        var option = AddOption("SPY");
        // Set the Option universe filter.
        option.SetFilter(x => x.IncludeWeeklys().Strikes(-5, 5).Expiration(7, 60));
        // Save a reference of the canonical symbol.
        _symbol = option.Symbol;
    }

    public override void OnData(Slice slice)
    {
        // Try to get the Option contracts within the Option chain.
        if (!Portfolio.Invested && slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            foreach (var contract in chain)
            {
                // Get the implied volatility and greeks of each Option contract.
                var iv = contract.ImpliedVolatility;
                var greeks = contract.Greeks;
                var delta = greeks.Delta;
                var gamma = greeks.Gamma;
                var vega = greeks.Vega;
                var theta = greeks.Theta;
                var rho = greeks.Rho;
            }

            // Invest in a straddle strategy to capitalize the volatility
            // Trade the ATM options with the closest expiry after 7 days.
            var expiry = chain.Min(x => x.Expiry);
            var selected = chain.Where(x => x.Expiry == expiry)
                .MinBy(x => Math.Abs(x.Greeks.Delta - 0.5m));
            var optionStrategy = OptionStrategies.Straddle(selected.Symbol.Canonical, selected.Strike,
selected.Expiry);
            Sell(optionStrategy, 1);
        }
    }
}
```

Example 5: Get Asset Sentiment Values

The following example gets sentiment values from the Brain Sentiment Indicator dataset. It simply buys AAPL when the sentiment is positive and liquidates the position when the sentiment is negative.

```C#
public class HandlingSecuritiesDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _dataset7DaySymbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Seed the price of AAPL with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Add the 7-day sentiment data for AAPL.
        _symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _dataset7DaySymbol = AddData<BrainSentimentIndicator7Day>(_symbol).Symbol;
    }

    public override void OnData(Slice slice)
    {
        // Check if the current slice contains the 7-day sentiment data.
        if (slice.ContainsKey(_dataset7DaySymbol))
        {
            var dataPoint = slice[_dataset7DaySymbol];
            // Log the sentiment value.
            Log($"{_dataset7DaySymbol} sentiment at {slice.Time}: {dataPoint.Sentiment}");

            // Invest if the sentiment score is above 0, which indicates positive sentiment.
            if (dataPoint.Sentiment > 0m)
            {
                MarketOrder(_symbol, 1);
            }
            // Liquidate otherwise.
            else if (dataPoint.Sentiment < 0m && Portfolio[_symbol].Invested)
            {
                Liquidate(_symbol);
            }
        }
    }
}
```

Example 6: Market Session as History

Using `Session` instead of history requests or rolling window to cache the last 252 OHLCV bars, the following algorithm buys SPY accordingly when the market opens above the 252-day max of closing prices.

C#

```
public class MarketSessionAsDailyHistory : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        var equity = AddEquity("SPY");

        // Increase the session lookback to 252 from default (0)
        // since our strategy break out the 252-day max of closing prices
        equity.Session.Size = 252;
        // Warm up with daily data for speed
        SetWarmUp(equity.Session.Size, Resolution.Daily);

        Schedule.On(DateRules.EveryDay("SPY"), TimeRules.AfterMarketOpen("SPY", 1), () =>
        {
            // Break-out strategy. If the market opens above the 252-day max of closing prices,
            // allocate all in SPY. Liquidate if it opens below the mean.

            var session = Securities["SPY"].Session;
            // The session is ready when the number of samples is greater than the size (252)
            if (IsWarmingUp || !session.IsReady)
                return;

            var closes = session.OrderBy(x => x.EndTime).Select(x => x.Close);
            var open = session.Open;
            var max = closes.Max();
            var mean = closes.Average();

            // Invest all in SPY if the open break out the 252-day max and liquidate below the mean
            if (open > max)
                SetHoldings("SPY", 1, tag: $"_open={open:F2}, _max={max:F2}");
            if (open < mean)
                Liquidate(symbol: "SPY", tag:$"_open={open:F2}, _mean={mean:F2}");
        });
    }
}
```

Securities

Market Hours

Introduction

The `Exchange` property of a `Security` object contains information about the exchange that lists the security, including its time zone and the trading days and hours. To get the exchange market hours, use the `Exchange.Hours` property on a `Security` object.

```
C#

// Get properties of the exchange that lists the security.
var securityExchangeHours = Securities["SPY"].Exchange.Hours;
```

Market Status

To check if the exchange is open at a specific time, call the `IsOpen` method.

```
C#

// Check if the exchange is currently open.
var isOpenNow = securityExchangeHours.IsOpen(Time, extendedMarketHours: false);

// Check if the exchange is open at any point in time over a specific interval.
var isOpen = securityExchangeHours.IsOpen(Time, Time.AddDays(1), extendedMarketHours: false);
```

To check if the exchange is open on a specific date, call the `IsDateOpen` method.

```
C#

// Check if the exchange is open at some time today.
var isOpen = securityExchangeHours.IsDateOpen(Time, extendedMarketHours: true);
```

To get the next market open time, call the `GetNextMarketOpen` method.

```
C#

// Get the next market open time.
var marketOpenTime = securityExchangeHours.GetNextMarketOpen(Time, extendedMarketHours: false);
```

To get the next market close time, call the `GetNextMarketClose` method.

```
C#

// Get the next market close time.
var marketCloseTime = securityExchangeHours.GetNextMarketClose(Time, extendedMarketHours: false);
```

To get the previous trading day, call the `GetPreviousTradingDay` method.

```
C#

// Get the previous trading day.
var day = securityExchangeHours.GetPreviousTradingDay(Time);
```

To get the next trading day, call the `GetNextTradingDay` method.

```
// Get the next trading day.
var day = securityExchangeHours.GetNextTradingDay(Time);
```

Extended Market Hours

By default, your security subscriptions only cover regular trading hours. To subscribe to pre and post-market trading hours for a specific asset, enable the `extendedMarketHours` argument when you create the security subscription.

```
AddEquity("SPY", extendedMarketHours: true);
```

You only receive extended market hours data if you create the subscription with minute, second, or tick resolution. If you create the subscription with daily or hourly resolution, the bars only reflect the regular trading hours.

Properties

The `SecurityExchange` object has the following attributes:

Examples

The follow examples demonstrate some common practices for trading with market hours data.

Example 1: Only Within 1 Hour Of Market Open And Close

Buying SPY shortly after market holidays or weekend breaks can be a profitable strategy. Historically, the market has tended to experience positive returns in the trading sessions immediately following extended market closures. This is often attributed to the release of pent-up demand and information during the downtime. By positioning into SPY at the open or within the first few trading hours after a holiday or weekend, investors may be able to capture this post-break bounce in the broader market.

```
public class MarketHourExampleAlgorithm : QCAlgorithm
{
    private Equity _spy;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        _spy = AddEquity("SPY", Resolution.Hour);
    }

    public override void OnData(Slice slice)
    {
        // Check if the previous day is holiday
        if (_spy.Exchange.Hours.Holidays.Contains(Time.Date.AddDays(-1)))
        {
            // Buy if it is a holiday a day before
            SetHoldings(_spy.Symbol, 1);
        }
        else if (Portfolio[_spy.Symbol].Invested)
        {
            // Liquidate otherwise if any open position
            Liquidate(_spy.Symbol);
        }
    }
}
```

Example 2: Pre-market Short Limit Order

A pre-market short limit order is a type of order placed before the regular trading session begins. It allows a trader to take a short position at a price higher than the current market price, with the intention of selling the position shortly after the market

opens and profiting from the expected price decline. This strategy is often used by traders who anticipate a decline in the asset's value and want to capitalize on it.

```C#
public class MarketHourExampleAlgorithm : QCAlgorithm
{
    private Equity _spy;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add SPY data with extended market hours.
        _spy = AddEquity("SPY", extendedMarketHours: true);
    }

    public override void OnData(Slice slice)
    {
        if (!slice.QuoteBars.ContainsKey(_spy.Symbol)) return;
        // During pre-market hours, place an unmarketable limit order.
        if (!_spy.Invested &&
            IsBeforeMarketOpenExtendedHours(slice.Time) &&
            Transactions.GetOpenOrderTickets(_spy.Symbol).Count() == 0)
        {
            LimitOrder(_spy.Symbol, -10, _spy.BidPrice * 1.005m);
        }
        // Check if the normal market is opening and any position
        else if (_spy.Invested && _spy.Exchange.Hours.IsOpen(slice.Time, false))
        {
            Liquidate(_spy.Symbol);
        }
    }

    private bool IsBeforeMarketOpenExtendedHours(DateTime time)
    {
        return (
            // It's currently extended market hours.
            _spy.Exchange.Hours.IsOpen(time, true) &&
            // The next market open for regular trading hours is today.
            _spy.Exchange.Hours.GetNextMarketOpen(time, false).Day == time.Day
        );
    }
}
```

However, it is important to note that pre-market trading can be more volatile and riskier than regular trading, and the intended price may not be achieved on this strategy.

Example 3: Last Hour Trades

Establishing a scheduled trading event in the hour before market close can be an effective strategy. This allows the trader to analyze market conditions, identify potential trade opportunities, and prepare orders ahead of the close. By emitting their trading insights and planned actions to the market in this critical final hour, the trader can influence other market participants and potentially gain an edge.

C#

```
public class MarketHourExampleAlgorithm : QCAlgorithm
{
    private Equity _spy;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        _spy = AddEquity("SPY");

        // Set up a portfolio construction model to handle insights for trading
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel(Expiry.EndOfDay));

        // Set up a scheduled event on 1-hour before market close to open position
        Schedule.On(
            DateRules.EveryDay(_spy.Symbol),
            TimeRules.BeforeMarketClose(_spy.Symbol, 60),
            LastHourTrade
        );
    }

    private void LastHourTrade()
    {
        // Get the time of next market close
        var marketCloseTime = _spy.Exchange.Hours.GetNextMarketClose(Time, extendedMarketHours: false);
        // Emit an insight that expires 1 minutye before the next market close
        EmitInsights(Insight.Price(_spy.Symbol, marketCloseTime.AddMinutes(-1), InsightDirection.Up));
    }
}
```

Securities

Filtering Data

Introduction

Unfiltered raw data can be faulty for a number of reasons, including invalid data entry. Moreover, high-frequency traders can deploy bait-and-switch strategies by submitting bait orders to deceive other market participants, making raw data noisy and untradeable. To avoid messing up with our trading logic and model training, you can filter out suspicious raw data with a data filter.

Set Models

To set a data filter for a security, call the `SetDataFilter` property on the `Security` object.

```
C#

// Use the SetDataFilter method to use the SecurityDataFilter on the SPY ETF data.
var spy = AddEquity("SPY");
spy.SetDataFilter(new SecurityDataFilter());
```

You can also set the data filter model in a security initializer . If your algorithm has a universe, use the security initializer technique. In order to initialize single security subscriptions with the security initializer, call `AddSecurityInitializer` before you create the subscriptions.

```
C#

public class AddSecurityInitializerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the security initializer to set models of assets.
        AddSecurityInitializer(CustomSecurityInitializer);
    }

    private void CustomSecurityInitializer(Security security)
    {
        // Overwrite some of the reality models
        security.SetDataFilter(new SecurityDataFilter());
    }
}
```

Default Behavior

The following table shows the default data filter for each security type:

Security Type	Default Filter
`Equity`	EquityDataFilter
`Option`	OptionDataFilter
`Forex`	ForexDataFilter
`Index`	IndexDataFilter
`Cfd`	CfdDataFilter
Others	SecurityDataFilter

None of the preceding filters filter out any data.

Model Structure

Data filtering models should implement the `ISecurityDataFilter` interface. Extensions of the `ISecurityDataFilter` interface must implement the `Filter` method, which receives `Security` and `BaseData` objects and then returns a `boolean` object that represents if the data point should be filtered out.

```C#
// Include or exclude a data point from the algorithm.
public class MyDataFilter : ISecurityDataFilter
{
    public override bool Filter(Security vehicle, BaseData data)
    {
        return true;
    }
}
```

Examples

The following examples demonstrate some common practices for filtering data.

Example 1: Filter Out Outliers

When analyzing high-frequency price data, it's important to filter out potential outliers and anomalies that may skew the analysis. One effective method is to use simple moving average (SMA) and standard deviation indicators to identify ticks that significantly deviate from the short-term trend. By comparing each tick to the indicator values, you can flag any data points that fall outside a threshold (for example, three standard deviations). This filtration process removes suspicious or erroneous price information from entering your algorithm, ensuring a cleaner dataset for trading.

```
C#
public class CustomDataFilterAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 3);
        dynamic equity = AddEquity("AAPL", Resolution.Tick);
        // Create the indicators.
        equity.Sma = SMA(equity.Symbol, 100);
        equity.Std = IndicatorExtensions.Of(new StandardDeviation(100), equity.Sma, true);
        // Set the data filter.
        equity.SetDataFilter(new CustomDataFilter());
    }
}

class CustomDataFilter : SecurityDataFilter
{
    public CustomDataFilter() : base() { }
    public override bool Filter(Security vehicle, BaseData data)
    {
        // Wait until the indicators are ready.
        var security = vehicle as dynamic;
        if (!(security.Sma.IsReady && security.Std.Window.IsReady))
        {
            return true; // Keep the data point.
        }
        // Check if the current value is within 3 standard deviations of the mean.
        // Return true (keep) or false (discard).
        var sma = security.Sma.Current.Value;
        var std = security.Std.Current.Value;
        return sma - 3m*std <= data.Value && data.Value <= sma + 3m*std;
    }
}
```

Example 2: Filter Out Major Exchanges

When you trade illiquid financial instruments, it can be advantageous to focus on the BATS exchange since its quote data may not fully reflect the fair market value. Due to the lower trading volume and visibility of BATS, the quotes there may lag behind the true value of illiquid assets. A carefully designed algorithm can analyze the BATS feed to identify situations where the quotes appears to be undervalued compared to the asset's intrinsic worth. By executing trades to capture this disconnect, rather than arbitraging between exchanges, you may be able to profit from the market inefficiencies present in the less liquid instrument. The following example demonstrates how to only consume data from the BATS exchange:

```
C#
public class BatsDataFilterAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 3);
        dynamic equity = AddEquity("AAPL", Resolution.Tick);
        // Set the data filter.
        equity.SetDataFilter(new BatsDataFilter());
    }
}

class BatsDataFilter : SecurityDataFilter
{
    public BatsDataFilter() : base() { }
    public override bool Filter(Security vehicle, BaseData data)
    {
        // Get the tick object.
        var tick = data as Tick;
        // Return true (keep) or false (discard).
        return tick != null && tick.Exchange == Exchange.BATS;
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
CustomSecurityDataFilterRegressionAlgorithm.cs C# TickDataFilteringAlgorithm.cs C#

Securities

Asset Classes

You can trade any of the following asset classes. Click one to learn more.

US Equity

Equity Options

Crypto

Crypto Futures

Forex

Futures

Future Options

Index

Index Options

CFD

India Equity

See Also

Universes
Importing Data

Asset Classes

US Equity

US Equities represent partial ownership in a US corporation.

Requesting Data

Handling Data

Corporate Actions

Corporate Fundamentals

Shorting

Data Preparation

Market Hours

See Also

BasicTemplateAlgorithm.py
BasicTemplateAlgorithm.cs

US Equity

Requesting Data

Introduction

Request US Equity data in your algorithm to receive a feed of asset prices in the `OnData` method.

Every stock traded in the US Securities Information Processors (SIP) CTA/UTP feed is available. Over-the-Counter (OTC) trades are not included. For more information about the specific dataset we use for backtests, see the US Equities dataset listing .

To trade US Equities live, you can use the QuantConnect data provider or one of the brokerage data providers .

Create Subscriptions

To create an Equity subscription, in the `Initialize` method, call the `AddEquity` method. The `AddEquity` method returns an `Equity` object, which contains a `Symbol` property. Save a reference to the `Symbol` so you can use it in `OnData` to access the security data in the `Slice` .

```
C#
_symbol = AddEquity("SPY").Symbol;
```

The `AddEquity` method creates a subscription for a single Equity asset and adds it to your user-defined universe. To create a dynamic universe of Equities, add an Equity universe or an Equity Universe Selection model .

To view the supported assets in the US Equities dataset, see the Data Explorer . It doesn't include Over-the-Counter (OTC) stocks.

Resolutions

The following table shows the available resolutions and data formats for Equity subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`			✓	✓
`Second`	✓	✓		
`Minute`	✓	✓		
`Hour`	✓			
`Daily`	✓			

The default resolution for Equity subscriptions is `Resolution.Minute` . To change the resolution, pass a `resolution` argument to the `AddEquity` method.

```
C#
_symbol = AddEquity("SPY", Resolution.Daily).Symbol;
```

To create custom resolution periods, see Consolidating Data .

Asset Primary Exchange

When a stock like Apple is listed, it's listed on Nasdaq. The open auction tick on Nasdaq is the price that's used as the official open of the day. NYSE, BATS, and other exchanges also have opening auctions, but the only official opening price for Apple is the opening auction on the exchange where it was listed.

Supported Markets

LEAN groups all of the US Equity exchanges under `Market.USA` . In live mode, the brokerage routes the orders to the exchange that provides the best price.

To set the market for a security, pass a `market` argument to the `AddEquity` method.

```
C#
_symbol = AddEquity("SPY", market: Market.USA).Symbol;
```

The brokerage models have a default market for each asset class. If you set a brokerage model, you may not need to specify the market to use.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
C#
AddEquity("SPY", fillForward: false);
```

Margin and Leverage

LEAN models buying power and margin calls to ensure your algorithm stays within the margin requirements. In backtests, the default leverage for margin accounts is 2x leverage and leverage is not available for cash accounts. To change the amount of leverage you can use for a security, pass a `leverage` argument to the `AddEquity` method.

```
C#
_symbol = AddEquity("SPY", leverage: 3).Symbol;
```

In live trading, the brokerage determines how much leverage you may use. For more information about the leverage they provide, see Brokerages .

Extended Market Hours

By default, your security subscriptions only cover regular trading hours. To subscribe to pre and post-market trading hours for a specific asset, enable the `extendedMarketHours` argument when you create the security subscription.

```
C#
AddEquity("SPY", extendedMarketHours: true);
```

You only receive extended market hours data if you create the subscription with minute, second, or tick resolution. If you create the subscription with daily or hourly resolution, the bars only reflect the regular trading hours.

To view the schedule of regular and extended market hours, see Market Hours .

Data Normalization

The data normalization mode defines how historical data is adjusted for corporate actions . The data normalization mode affects the data that LEAN passes to `OnData` and the data from history requests . By default, LEAN adjusts US Equity data for splits and dividends to produce a smooth price curve, but the following data normalization modes are available for Equity subscriptions:

If you use `Adjusted` , `SplitAdjusted` , or `TotalReturn` , we use the entire split and dividend history to adjust historical prices. This process ensures you get the same adjusted prices, regardless of the backtest end date.

To set the data normalization mode for a security, pass a `dataNormalizationMode` argument to the `AddEquity` method.

```
C#
_symbol = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;
```

The `ScaledRaw` data normalization is only for history requests . When you use `ScaledRaw` , we use the split and dividend history before the algorithm's current time to adjust historical prices. The `ScaledRaw` data normalization model enables you to warm up indicators with adjusted data when you subscribe to `Raw` security data.

Properties

The `AddEquity` method returns an `Equity` object, which have the following properties:

Examples

The following examples demonstrate some common practices for requesting US Equity data.

Example 1: Add Multiple Assets

The following algorithm adds multiple US Equities with extended market hours data. Extended market hours data enables you to monitor price movements during pre-market and after-hours trading sessions.

```
C#
public class IndexDemoAlgorithm : QCAlgorithm
{
    private List<Equity> _equities = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Create a list of tickers you want to add.
        var tickers = new string[] { "AAPL", "TSLA", "GOOG" };
        // Add extended market hours data for each US Equity.
        _equities = tickers.Select(ticker => AddEquity(ticker, extendedMarketHours: true)).ToList();
    }
}
```

Example 2: Import External Trade Signals

Say you source trading signals from outside your algorithm and organize them into a CSV file on Dropbox with the following structure:

```
#ticker,source1,source2,source3,source4,source5
AAPL,1.25,1.10,1.32,1.13,1.21
AMZN,1.15,1.05,1.07,1.11,1.03
GOOG,0.97,0.99,0.98,1.04,0.95
META,1.04,1.03,1.10,1.04,0.98
MSFT,0.92,0.96,0.98,1.01,0.98
```

The following algorithm uses a Scheduled Event to periodically pull the CSV file from Dropbox , parse its signals, and place trades. To determine the target weight for each asset, it takes the mean score from all the sources. If the mean score is greater than 1, it allocates 20% of the portfolio to the asset. Otherwise, it allocates -20% to the asset.

C#

```
public class USEquityDemoAlgorithm : QCAlgorithm
{
    private readonly Dictionary<string, decimal> _signalByTicker = [];

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        Settings.SeedInitialPrices = true;

        // Add the US Equities.
        foreach (var ticker in "AAPL,AMZN,GOOG,META,MSFT".Split(","))
        {
            AddEquity(ticker, Resolution.Daily);
            _signalByTicker.Add(ticker, 1);
        }
        // Set up a Scheduled Event to download the data file and place orders.
        Schedule.On(DateRules.MonthStart(), TimeRules.Midnight, DownloadDataAndPlaceOrders);
    }

    private void DownloadDataAndPlaceOrders()
    {
        // Download the file from DropBox.
        var fileContent = Download("<DropBoxUrl>&dl=1");

        // Convert the file content into trading signals.
        foreach (var line in fileContent.Split(['\n', '\r'], StringSplitOptions.RemoveEmptyEntries).Skip(1))
        {
            var values = line.Split(',');
            _signalByTicker[values[0]] = values.Skip(1).Select(decimal.Parse).ToArray().Average();
        }

        // Place orders according to the signals.
        var targets = _signalByTicker.Select(kvp => new PortfolioTarget(kvp.Key, kvp.Value >= 1m ? 0.2m : -0.2m));
        SetHoldings([.. targets], true);
    }
}
```

US Equity

Handling Data

Introduction

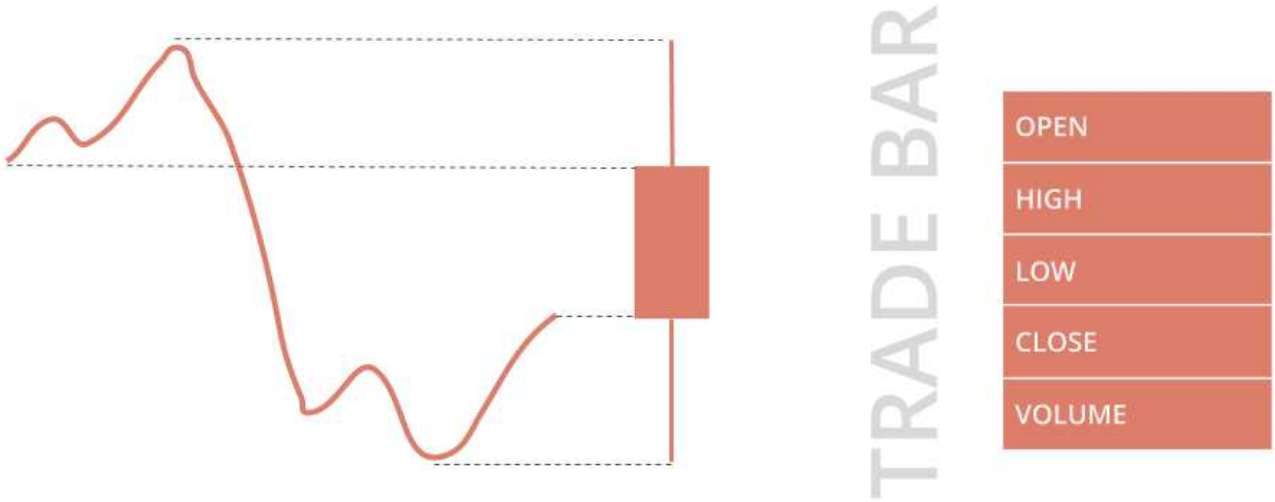
LEAN passes the data you request to the `OnData` method so you can make trading decisions. The default `OnData` method accepts a `Slice` object, but you can define additional `OnData` methods that accept different data types. For example, if you define an `OnData` method that accepts a `TradeBar` argument, it only receives `TradeBar` objects. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `TradeBars` `DataDictionary` is made up of `TradeBar` objects. To access individual data points in the dictionary, you can index the dictionary with the security ticker or `Symbol` , but we recommend you use the `Symbol` .

To view the resolutions that are available for US Equity data, see Resolutions .

Trades

`TradeBar` objects are price bars that consolidate individual trades from the exchanges. They contain the open, high, low, close, and volume of trading activity over a period of time.



To get the `TradeBar` objects in the `Slice` , index the `Slice` or index the `Bars` property of the `Slice` with the security `Symbol` . If the security doesn't actively trade or you are in the same time step as when you added the security subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your security before you index the `Slice` with the security `Symbol` .

```C#
public override void OnData(Slice slice)
{
    // Check if the symbol is contained in TradeBars object
    if (slice.Bars.ContainsKey(_symbol))
    {
        // Obtain the mapped TradeBar of the symbol
        var tradeBar = slice.Bars[_symbol];
    }
}
```

You can also iterate through the `TradeBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `TradeBar` objects.

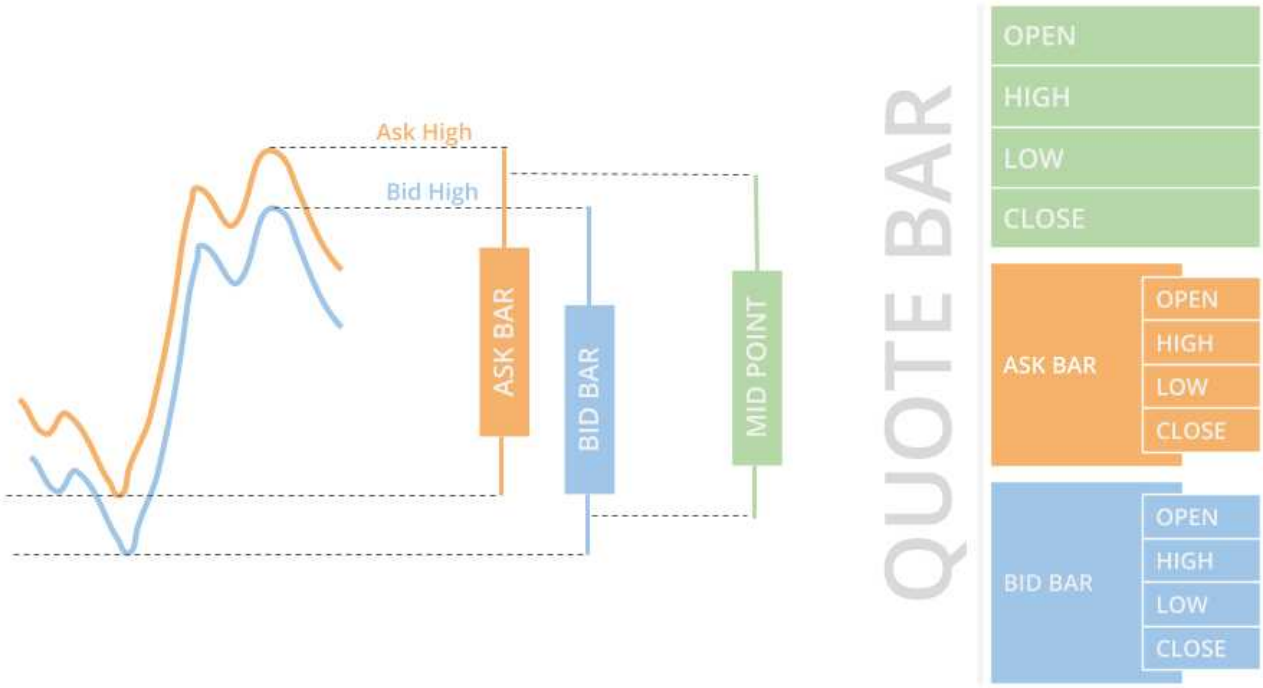
```C#
public override void OnData(Slice slice)
{
    // Iterate all received Symbol-TradeBar key-value pairs
    foreach (var kvp in slice.Bars)
    {
        var symbol = kvp.Key;
        var tradeBar = kvp.Value;
        var closePrice = tradeBar.Close;
    }
}
```

`TradeBar` objects have the following properties:

We adjust the daily open and close price of bars to reflect the official auction prices .

Quotes

`QuoteBar` objects are bars that consolidate NBBO quotes from the exchanges. They contain the open, high, low, and close prices of the bid and ask. The `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` object are the mean of the respective bid and ask prices. If the bid or ask portion of the `QuoteBar` has no data, the `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` copy the values of either the `Bid` or `Ask` instead of taking their mean.



To get the QuoteBar objects in the Slice , index the QuoteBars property of the Slice with the security Symbol . If the security doesn't actively get quotes or you are in the same time step as when you added the security subscription, the Slice may not contain data for your Symbol . To avoid issues, check if the Slice contains data for your security before you index the Slice with the security Symbol .

```C#
public override void OnData(Slice slice)
{
    // Check if the symbol is contained in QuoteBars object
    if (slice.QuoteBars.ContainsKey(_symbol))
    {
        // Obtain the mapped QuoteBar of the symbol
        var quoteBar = slice.QuoteBars[_symbol];
    }
}
```

You can also iterate through the QuoteBars dictionary. The keys of the dictionary are the Symbol objects and the values are the QuoteBar objects.

```C#
public override void OnData(Slice slice)
{
    // Iterate all received Symbol-QuoteBar key-value pairs
    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        var askPrice = quoteBar.Ask.Close;
    }
}
```

QuoteBar objects let LEAN incorporate spread costs into your simulated trade fills to make backtest results more realistic.

QuoteBar objects have the following properties:

Ticks

Tick objects represent a single trade or quote at a moment in time. A trade tick is a record of a transaction for the security. A quote tick is an offer to buy or sell the security at a specific price.

Trade ticks have a non-zero value for the Quantity and Price properties, but they have a zero value for the BidPrice , BidSize , AskPrice , and AskSize properties. Quote ticks have non-zero values for BidPrice and BidSize properties or have non-zero values for AskPrice and AskSize properties. To check if a tick is a trade or a quote, use the TickType property.

In backtests, LEAN groups ticks into one millisecond buckets. In live trading, LEAN groups ticks into ~70-millisecond buckets. To get the Tick objects in the Slice , index the Ticks property of the Slice with a Symbol . If the security doesn't actively trade or you are in the same time step as when you added the security subscription, the Slice may not contain data for your Symbol . To avoid issues, check if the Slice contains data for your security before you index the Slice with the security Symbol .

```C#
public override void OnData(Slice slice)
{
    if (slice.Ticks.ContainsKey(_symbol))
    {
        var ticks = slice.Ticks[_symbol];
        foreach (var tick in ticks)
        {
            var price = tick.Price;
        }
    }
}
```

You can also iterate through the `Ticks` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `List<Tick>` objects.

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Ticks)
    {
        var symbol = kvp.Key;
        var ticks = kvp.Value;
        foreach (var tick in ticks)
        {
            var price = tick.Price;
        }
    }
}
```

Tick data is raw and unfiltered, so it can contain bad ticks that skew your trade results. For example, some ticks come from dark pools, which aren't tradable. We recommend you only use tick data if you understand the risks and are able to perform your own online tick filtering.

`Tick` objects have the following properties:

If we detect a tick that may be suspicious , we set its `Suspicious` flag to true.

Other Data Formats

For more information about data formats available for US Equities, see Corporate Actions .

Examples

The following examples demonstrate some common practices for handling US Equity data.

Example 1: Various Data Resolutions and Formats

The following algorithm handles three data resolutions (Daily, Minute, and Tick) and three data formats (TradeBar, QuoteBar, and Tick). The Equity subscriptions provide the ticks and daily TradeBar objects. To create the 5-minute QuoteBar objects, it consolidates the AAPL ticks.

The algorithm enters a long position in AAPL when the following conditions are met:

- The 16-day Fractal Adaptive Moving Average (FRAMA) of QQQ is increasing, which indicates the trend is up.
- The Relative Strength Index (RSI) of AAPL is above 70, which indicates AAPL has been rising from large buying pressure.

The algorithm enters a short position in AAPL when the following conditions are met:

- The 16-day FRAMA of QQQ is rising, which indicates the trend is down.
- The RSI of AAPL is above, which indicates AAPL has been declining from large selling pressure.

The algorithm liquidates the portfolio when the FRAMA changes direction.

C#

```
public class USEquityExampleAlgorithm : QCAlgorithm
{
    private dynamic _qqq, _aapl;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 4);
        SetEndDate(2024, 9, 10);

        // Add QQQ and AAPL data.
        _qqq = AddEquity("QQQ", Resolution.Daily);
        _aapl = AddEquity("AAPL", Resolution.Tick);

        // Create the indicators.
        _qqq.Frama = FRAMA(_qqq.Symbol, 16);
        _aapl.Rsi = new RelativeStrengthIndex(10);

        // Create a 5-minute consolidator to aggregate AAPL tick data into quote bars.
        var consolidator = new TickQuoteBarConsolidator(TimeSpan.FromMinutes(5));
        // Subscribe the consolidator for automatic updates.
        SubscriptionManager.AddConsolidator(_aapl.Symbol, consolidator);
        // Attach an event handler to update the RSI indicator with the consolidated bars.
        consolidator.DataConsolidated += (sender, bar) => _aapl.Rsi.Update(bar.EndTime, bar.Close);

        // Warm up the indicators.
        WarmUpIndicator(_qqq.Symbol, _qqq.Frama);
        var history = History<QuoteBar>(_aapl.Symbol, 10, Resolution.Minute);
        foreach (var bar in history)
        {
            _aapl.Rsi.Update(bar.EndTime, bar.Close);
        }
    }

    public override void OnData(Slice slice)
    {
        // Ensure AAPL ticks are in the current slice.
        if (!slice.Ticks.ContainsKey(_aapl.Symbol))
        {
            return;
        }

        // Get the indicator values.
        var frama = _qqq.Frama.Current.Value;
        var prevFrama = _qqq.Frama.Previous.Value;
        var rsi = _aapl.Rsi.Current.Value;

        // Check if we have an AAPL position.
        if (_aapl.Holdings.Invested)
        {
            // If there is divergence between our position and the Frama direction, exit the position.
            if ((_aapl.Holdings.IsLong && frama < prevFrama) || (_aapl.Holdings.IsShort && frama > prevFrama))
            {
                Liquidate(_aapl.Symbol);
            }
        }
        // Look for a long entry.
        else if (frama > prevFrama && rsi >= 70)
        {
            SetHoldings(_aapl.Symbol, 0.5);
        }
        // Look for a short entry.
        else if (frama < prevFrama && rsi <= 30)
        {
            SetHoldings(_aapl.Symbol, -0.5);
        }
    }
}
```

US Equity

Corporate Actions

Introduction

US Equity subscriptions provide notifications for splits, dividends, symbol changes, and delistings.

Splits

When a company does a stock split, the number of shares each shareholder owns increases and the price of each share decreases. When a company does a reverse stock split, the number of shares each shareholder owns decreases and the price of each share increases. A company may perform a stock split or a reverse stock split to adjust the price of their stock so that more investors trade it and the liquidity increases.

When a stock split or reverse stock split occurs for an Equity in your algorithm, LEAN sends a `Split` object to the `OnData` method. You receive `Split` objects when a split is in the near future and when it occurs. To know if the split occurs in the near future or now, check the `Type` property.

If you backtest without the `Raw` data normalization mode , the splits are factored into the price and volume. If you backtest with the `Raw` data normalization mode or trade live, when a split occurs, LEAN automatically adjusts your positions based on the `SplitFactor` . If the post-split quantity isn't a valid lot size , LEAN credits the remaining value to your cashbook in your account currency. If you have indicators or consolidators in your algorithm, reset and warm-up them up with `ScaledRaw` data when splits occur so that the data in your indicators and consolidators account for the price adjustments that the splits cause. For examples, see Reset Indicators or Reset Consolidators .

To get the `Split` objects, index the `Splits` object with the security `Symbol` . The `Splits` object may not contain data for your `Symbol` . To avoid issues, check if the `Splits` object contains data for your security before you index it with the security `Symbol` .

```C#
public override void OnData(Slice slice)
{
    if (slice.Splits.ContainsKey(_symbol))
    {
        var split = slice.Splits[_symbol];
    }
}

public override void OnSplits(Splits splits)
{
    if (splits.ContainsKey(_symbol))
    {
        var split = splits[_symbol];
    }
}
```

You can also iterate through the `Splits` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `Split` objects.

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Splits)
    {
        var symbol = kvp.Key;
        var split = kvp.Value;
    }
}

public override void OnSplits(Splits splits)
{
    foreach (var kvp in splits)
    {
        var symbol = kvp.Key;
        var split = kvp.Value;
    }
}
```

To get historical splits, make a history request with the `Split` type.

```C#
var splits = History<Split>(symbol, TimeSpan.FromDays(4*365));
```

LEAN stores the data for stock splits in factor files. To view some example factor files, see the LEAN GitHub repository . In backtests, your algorithm receives `Split` objects at midnight. In live trading, your algorithm receives `Split` objects when the factor files are ready.

If you hold an Option contract for an underlying Equity when a split occurs, LEAN closes your Option contract position.

If a split event occurs before your order is filled, the unfilled portion of the order is adjusted automatically, where its quantity is multiplied by the split factor and the limit/stop/trigger price (if any) is divided by the split factor.

`Split` objects have the following properties:

Dividends

A dividend is a payment that a company gives to shareholders to distribute profits. When a dividend payment occurs for an Equity in your algorithm, LEAN sends a `Dividend` object to the `OnData` method.

If you backtest with the `Adjusted` or `TotalReturn` data normalization mode, the dividends are factored into the price. If you backtest with the other data normalization modes or trade live, when a dividend payment occurs, LEAN automatically adds the payment amount to your cashbook. If you have indicators or consolidators in your algorithm, reset and warm-up them up with `ScaledRaw` data when dividend payments occur so that the data in your indicators and consolidators account for the price adjustments that the dividend causes. For examples, see Reset Indicators or Reset Consolidators .

To get the `Dividend` objects, index the `Dividends` object with the security `Symbol` . The `Dividends` object may not contain data for your `Symbol` . To avoid issues, check if the `Dividends` object contains data for your security before you index it with the security `Symbol` .

```C#
public override void OnData(Slice slice)
{
    if (slice.Dividends.ContainsKey(_symbol))
    {
        var dividend = slice.Dividends[_symbol];
    }
}

public override void OnDividends(Dividends dividends)
{
    if (dividends.ContainsKey(_symbol))
    {
        var dividend = dividends[_symbol];
    }
}
```

You can also iterate through the `Dividends` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `Dividend` objects.

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Dividends)
    {
        var symbol = kvp.Key;
        var dividend = kvp.Value;
    }
}

public override void OnDividends(Dividends dividends)
{
    foreach (var kvp in dividends)
    {
        var symbol = kvp.Key;
        var dividend = kvp.Value;
    }
}
```

To get historical dividends, make a history request with the `Dividend` type.

```C#
var dividends = History<Dividend>(symbol, TimeSpan.FromDays(4*365));
```

For a full example, see the DividendAlgorithm in the LEAN GitHub repository.

`Dividend` objects have the following properties:

Symbol Changes

The benefit of the `Symbol` class is that it always maps to the same security, regardless of their trading ticker. When a company changes its trading ticker, LEAN sends a `SymbolChangedEvent` to the `OnData` method.

To get the `SymbolChangedEvent` objects, index the `SymbolChangedEvents` object with the security `Symbol` . The `SymbolChangedEvents` object may not contain data for your `Symbol` . To avoid issues, check if the `SymbolChangedEvents` object contains data for your security before you index it with the security `Symbol` .

```C#
public override void OnData(Slice slice)
{
    if (slice.SymbolChangedEvents.ContainsKey(_symbol))
    {
        var symbolChangedEvent = slice.SymbolChangedEvents[_symbol];
    }
}

public override void OnSymbolChangedEvents(SymbolChangedEvents symbolChangedEvents)
{
    if (symbolChangedEvents.ContainsKey(_symbol))
    {
        var symbolChangedEvent = symbolChangedEvents[_symbol];
    }
}
```

You can also iterate through the `SymbolChangedEvents` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `SymbolChangedEvent` objects.

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.SymbolChangedEvents)
    {
        var symbol = kvp.Key;
        var symbolChangedEvent = kvp.Value;
    }
}

public override void OnSymbolChangedEvents(SymbolChangedEvents symbolChangedEvents)
{
    foreach (var kvp in symbolChangedEvents)
    {
        var symbol = kvp.Key;
        var symbolChangedEvent = kvp.Value;
    }
}
```

If you have an open order for a security when they change their ticker, LEAN cancels your order. To keep your order, in the `OnOrderEvent` method, get the quantity and `Symbol` of the cancelled order and submit a new order.

```C#
public override void OnOrderEvent(OrderEvent orderEvent)
{
    if (orderEvent.Status == OrderStatus.Canceled)
    {
        var ticket = Transactions.GetOrderTicket(orderEvent.OrderId);
        if (ticket.Tag.Contains("symbol changed event"))
        {
            Transactions.AddOrder(ticket.SubmitRequest);
        }
    }
}
```

LEAN stores the data for ticker changes in map files. To view some example map files, see the LEAN GitHub repository .

`SymbolChangedEvent` objects have the following properties:

Delistings

When a company is delisting from an exchange, LEAN sends a `Delisting` object to the `OnData` method. You receive `Delisting` objects when a delisting is in the near future and when it occurs. To know if the delisting occurs in the near future or now, check the `Type` property.

To get the `Delisting` objects, index the `Delistings` object with the security `Symbol` . The `Delistings` objects may not contain data for your `Symbol` . To avoid issues, check if the `Delistings` object contains data for your security before you index it with the security `Symbol` .

```
C#
public override void OnData(Slice slice)
{
    if (slice.Delistings.ContainsKey(_symbol))
    {
        var delisting = slice.Delistings[_symbol];
    }
}

public override void OnDelistings(Delistings delistings)
{
    if (delistings.ContainsKey(_symbol))
    {
        var delisting = delistings[_symbol];
    }
}
```

You can also iterate through the `Delistings` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `Delisting` objects.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Delistings)
    {
        var symbol = kvp.Key;
        var delisting = kvp.Value;
    }
}

public override void OnDelistings(Delistings delistings)
{
    foreach (var kvp in delistings)
    {
        var symbol = kvp.Key;
        var delisting = kvp.Value;
    }
}
```

The delist warning occurs on the final trading day of the stock to give you time to gracefully exit out of positions before LEAN automatically liquidates them.

```
C#
if (delisting.Type == DelistingType.Warning)
{
    // Liquidate with MarketOnOpenOrder on delisting warning
    var quantity = Portfolio[symbol].Quantity;
    if (quantity != 0)
    {
        MarketOnOpenOrder(symbol, -quantity);
    }
}
```

The `OnSecuritiesChanged` event notifies the algorithm when delisted assets are removed from the universe.

To improve iteration speed, LEAN removes delisted securities from the `Securities` primary collection. To access all the securities, iterate the `Securities.Total` property.

```
// Access all the securities.
foreach (var security in Securities.Total)
{

}

// Exclude delisted securities.
foreach (var security in Securities.Values)
{

}
```

For a full example, see the DelistingEventsAlgorithm in the LEAN GitHub repository.

`Delisting` objects have the following properties:

Live Trading Considerations

In backtesting, corporate actions occurs at midnight Eastern Time (ET). In live trading, the live data for corporate actions arrives at 6/7 AM ET, so that's when they occur.

Examples

The following examples demonstrate some common practices for US Equity corporate actions.

Example 1: Resetting Indicators on Split Events

The following algorithm adds raw TSLA data and uses it to update an Exponential Moving Average (EMA) indicator. When a split occurs, it resets and warms up the indicator with scaled raw data to ensure the indicator doesn't reflect price discontinuities.

```
public class USEquityCorporateActionExampleAlgorithm : QCAlgorithm
{
    private dynamic _equity;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add raw US Equity data.
        _equity = AddEquity("TSLA", Resolution.Daily, dataNormalizationMode: DataNormalizationMode.Raw);
        // Create an EMA indicator.
        _equity.Ema = EMA(_equity.Symbol, 50);
    }

    public override void OnSplits(Splits splits)
    {
        // Wait until the split has occurred.
        var split = splits[_equity.Symbol];
        if (split.Type != SplitType.SplitOccurred)
        {
            return;
        }
        // Reset the indicator.
        _equity.Ema.Reset();
        // Warm up the indicator with scaled raw data.
        var history = History<TradeBar>(
            _equity.Symbol, _equity.Ema.WarmUpPeriod, dataNormalizationMode: DataNormalizationMode.ScaledRaw
        );
        foreach (var bar in history)
        {
            _equity.Ema.Update(bar.EndTime, bar.Close);
        }
    }
}
```

Example 2: Reinvesting Dividend Payments

The following algorithm adds raw SPY data. It then holds the SPY and re-invests the dividend payments into additional shares.

C#

```
public class USEquityCorporateActionExampleAlgorithm : QCAlgorithm
{
    private Equity _spy;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add raw Equity data.
        _spy = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw);
    }

    public override void OnData(Slice slice)
    {
        // Buy and hold.
        if (!_spy.Holdings.Invested)
        {
            SetHoldings(_spy.Symbol, 0.5);
        }
    }

    public override void OnDividends(Dividends dividends)
    {
        // Get the total dividend payment amount.
        var totalDividendAmount = dividends[_spy.Symbol].Distribution * _spy.Holdings.Quantity;

        // Calculate the number of shares you can buy with the dividend payment.
        var quantity = Math.Floor(totalDividendAmount / _spy.Price);
        if (quantity > 0)
        {
            // Place an order to re-invest the dividends.
            MarketOrder(_spy.Symbol, quantity);
        }
    }
}
```

US Equity

Corporate Fundamentals

Introduction

Corporate fundamental data contains all the information on the underlying company of an Equity asset and the information in their financial statements. Since corporate data contains information not found in price and alternative data, adding corporate data to your trading strategies provides you with more information so you can make more informed trading decisions. Corporate fundamental data is available through the US Fundamental Data from Morningstar .

Direct Access

To get fundamental data for Equities in your algorithm, use the `Fundamentals` property of the `Equity` objects. The fundamental data represent the company's fundamentals for the current algorithm time.

```
C#
var fundamentals = Securities[_symbol].Fundamentals;
```

To get fundamental data for Equities, regardless of whether or not you have subscribed to them in your algorithm, call the `Fundamentals` method. If you pass one `Symbol` , the method returns a `Fundamental` object. If you pass a list of `Symbol` objects, the method returns a list of `Fundamental` objects. The fundamental data represents the corporate fundamentals for the current algorithm time.

```
C#
// Single asset
var ibm = QuantConnect.Symbol.Create("IBM", SecurityType.Equity, Market.USA);
var ibmFundamental = Fundamentals(ibm);

// Multiple assets
var nb = QuantConnect.Symbol.Create("NB", SecurityType.Equity, Market.USA);
var fundamentals = Fundamentals(new List<Symbol>{ nb, ibm }).ToList();
```

Data Availability

Some assets don't have fundamentals (for example, ETFs) and the Morningstar dataset doesn't provide fundamentals for all US Equities. To check if fundamental data is available for an asset, use the `HasFundamentalData` property.

```
C#
var hasFundamentalData = Securities[_symbol].Fundamentals.HasFundamentalData;
```

Object References

If you save a reference to the `Fundamentals` object or its properties, you can access the fundamental properties as they change over time.

```
C#
_fundamentals = Securities[_symbol].Fundamentals;
var earningRatios = _fundamentals.EarningRatios;
```

Universe Selection

You can access fundamental data in the selection function of your fundamental universe .

```
C#
public class MyUniverseAlgorithm : QCAlgorithm {
    public override void Initialize()
    {
        UniverseSettings.Asynchronous = true;
        AddUniverse(FundamentalFilterFunction);
    }

    private IEnumerable<Symbol> FundamentalFilterFunction(IEnumerable<Fundamental> fundamental)
    {
         return (from f in fundamental
                where f.HasFundamentalData
                select f.Symbol);
    }
}
```

Historical Data

To get historical fundamental data, call the `History` method. The return type depends on how you call the method.

```
C#
var ibm = QuantConnect.Symbol.Create("IBM", SecurityType.Equity, Market.USA);

// Fundamental objects
var fundamentalHistory = History<Fundamental>(ibm, TimeSpan.FromDays(30));

// Fundamentals objects for all US Equities (including delisted companies)
var fundamentalsHistory = History<Fundamentals>(TimeSpan.FromDays(30));

// Collection of Fundamental objects for all US Equities (including delisted companies)
var collectionHistory = History(_universe, 30, Resolution.Daily);
foreach (var fundamental in collectionHistory)
{
    // Cast to Fundamental is required
    var highestMarketCap = fundamental.OfType<Fundamental>().OrderByDescending(x => x.MarketCap).Take(5);
}
```

For more information about historical fundamental data, see Equity Fundamental Data .

Data Updates

The US Fundamental dataset only provides the originally-reported figures. If there was a mistake in reporting a figure, the data value isn't fixed later. In live trading, new fundamental data is available to your algorithms at approximately 6/7 AM Eastern Time (ET) each day. The majority of the corporate data update occurs once per month, but the financial ratios update daily. If there is no new data for a period of time, the previous data is filled forward.

Properties

The US Fundamentals dataset provides Fundamental objects. To filter Fundamental objects, you can use the `MorningstarSectorCode` , `MorningstarIndustryGroupCode` , and `MorningstarIndustryCode` enumeration values.

Fundamentals Attributes

Fundamentals objects have the following attributes:

MorningstarSectorCode Enumeration

Sectors are large super categories of data. To access the sector of an Equity, use the **MorningstarSectorCode** property.

```C#
filtered = fundamental.Where(x => x.AssetClassification.MorningstarSectorCode == MorningstarSectorCode.Technology);
```

The **MorningstarSectorCode** enumeration has the following members:

MorningstarIndustryGroupCode Enumeration

Industry groups are clusters of related industries which tie together. To access the industry group of an Equity, use the **MorningstarIndustryGroupCode** property.

```C#
filtered = fundamental.Where(x => x.AssetClassification.MorningstarIndustryGroupCode ==
MorningstarIndustryGroupCode.ApplicationSoftware);
```

The **MorningstarIndustryGroupCode** enumeration has the following members:

MorningstarIndustryCode Enumeration

Industries are the finest level of classification available and are the individual industries according to the Morningstar classification system. To access the industry group of an Equity, use the **MorningstarIndustryCode** property:

```C#
filtered = fundamental.Where(x => x.AssetClassification.MorningstarIndustryCode ==
MorningstarIndustryCode.SoftwareApplication);
```

The **MorningstarIndustryCode** enumeration has the following members:

Exchange Id Values

Exchange Id is mapped to represent the exchange that lists the Equity. To access the exchange Id of an Equity, use the **PrimaryExchangeID** property.

```C#
filtered = fundamental.Where(x => x.CompanyReference.PrimaryExchangeID == "NAS");
```

The exchanges are represented by the following string values:

String Representation	**Exchange**
NYS	New York Stock Exchange (NYSE)
NAS	NASDAQ
ASE	American Stock Exchange (AMEX)
TSE	Tokyo Stock Exchange
AMS	Amsterdam Internet Exchange
SGO	Santiago Stock Exchange
XMAD	Madrid Stock Exchange

ASX	Australian Securities Exchange
BVMF	B3 (stock exchange)
LON	London Stock Exchange
TKS	Istanbul Stock Exchange Settlement and Custody Bank
SHG	Shanghai Exchange
LIM	Lima Stock Exchange
FRA	Frankfurt Stock Exchange
JSE	Johannesburg Stock Exchange
MIL	Milan Stock Exchange
TAE	Tel Aviv Stock Exchange
STO	Stockholm Stock Exchange
ETR	Deutsche Boerse Xetra Core
PAR	Paris Stock Exchange
BUE	Buenos Aires Stock Exchange
KRX	Korea Exchange
SWX	SIX Swiss Exchange
PINX	Pink Sheets (OTC)
CSE	Canadian Securities Exchange
PHS	Philippine Stock Exchange
MEX	Mexican Stock Exchange
TAI	Taiwan Stock Exchange
IDX	Indonesia Stock Exchange
OSL	Oslo Stock Exchange
BOG	Colombia Stock Exchange
NSE	National Stock Exchange of India
HEL	Nasdaq Helsinki
MISX	Moscow Exchange
HKG	Hong Kong Stock Exchange

IST	Istanbul Stock Exchange
BOM	Bombay Stock Exchange
TSX	Toronto Stock Exchange
BRU	Brussels Stock Exchange
BATS	BATS Global Markets
ARCX	NYSE Arca
GREY	Grey Market (OTC)
DUS	Dusseldorf Stock Exchange
BER	Berlin Stock Exchange
ROCO	Taipei Exchange
CNQ	Canadian Trading and Quotation System Inc.
BSP	Bangko Sentral ng Pilipinas
NEOE	NEO Exchange

Examples

The following examples demonstrate some common practices for US Equity corporate fundamentals.

Example 1: Capturing the Overreaction to Financial Reports

The following algorithm is based on the hypothesis that investors overreact to new financial report releases. It buys TSLA stock on every new 10Q or 8K filing and then liquidates the position after seven days.

```C#
public class USEquityCorporateFundamentalsExampleAlgorithm : QCAlgorithm
{
    private Equity _equity;
    // Add a member to track the last financial statement filing date.
    private DateTime? _lastFilingDate = null;
    private DateTime _exitTime;

    public override void Initialize()
    {
        SetStartDate(2024,1,1);
        SetEndDate(2024,12,31);
        // Add Equity data.
        _equity = AddEquity("TSLA");
    }

    public override void OnData(Slice slice)
    {
        // Get the most recent filing date.
        var lastFilingDate = _equity.Fundamentals.FinancialStatements.FileDate.Value;

        // When new financial statements are released, buy TSLA.
        if (_lastFilingDate.HasValue && lastFilingDate != _lastFilingDate.Value)
        {
            SetHoldings(_equity.Symbol, 1);
            _exitTime = Time.AddDays(7);
        }

        // Liquidate the position after 1 week.
        if (_equity.Holdings.Invested && Time >= _exitTime)
        {
            Liquidate();
        }

        // Update the last filing date to identify future changes.
        _lastFilingDate = lastFilingDate;
    }
}
```

Example 2: Diversifying Across Sectors

The follow algorithm aims to reduce concentration risk in any one sector. It selects the 500 largest Equities at the start of each month and then forms a portfolio that gives equal weight to each sector and equal weight to the assets within each sector.

C#

```
public class USEquityCorporateFundamentalsExampleAlgorithm : QCAlgorithm
{
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2024,1,1);
        SetEndDate(2024,12,31);
        SetCash(10_000_000);
        // Define the universe settings.
        UniverseSettings.Resolution = Resolution.Daily;
        UniverseSettings.Schedule.On(DateRules.MonthStart("SPY"));
        // Add a universe of the 500 largest Equities.
        _universe = AddUniverse(coarse => coarse
            .OrderByDescending(x => x.MarketCap)
            .Take(500)
            .Select(x => x.Symbol)
        );
        // Create a Scheduled Event to rebalance the portfolio each month.
        Schedule.On(DateRules.MonthStart("SPY", 1), TimeRules.Midnight, Rebalance);
    }

    private void Rebalance()
    {
        // Wait until there are assets in the universe.
        if (_universe.Selected == null)
        {
            return;
        }
        // Group the universe constituents by sector.
        var symbolsBySector = new Dictionary<int, List<Symbol>>();
        foreach (var symbol in _universe.Selected)
        {
            var equity = Securities[symbol];
            if (equity.Price == 0)
            {
                continue;
            }
            var sectorCode = equity.Fundamentals.AssetClassification.MorningstarSectorCode;
            if (!symbolsBySector.ContainsKey(sectorCode))
            {
                symbolsBySector[sectorCode] = new List<Symbol>();
            }
            symbolsBySector[sectorCode].Add(symbol);
        }

        // Create targets to invest equally in each sector and invest equally in the assets within each sector.
        var targets = new List<PortfolioTarget>();
        var sectorCount = symbolsBySector.Count;
        foreach (var symbols in symbolsBySector.Values)
        {
            var weight = 1m / sectorCount / symbols.Count;
            targets.AddRange(symbols.Select(symbol => new PortfolioTarget(symbol, weight)));
        }

        // Place the trades.
        SetHoldings(targets, liquidateExistingHoldings: true);
    }
}
```

US Equity

Shorting

Introduction

In a regular long position, you buy shares and then sell them later to close the trade. In a short position, you borrow shares, sell them, and then buy the shares back later to close the trade. Short positions let you profit when a security's price decreases while you have the position open, but there are risks involved. For example, you can get a margin call and you may incur borrowing costs.

Holdings Accounting

If you have an open long position, your portfolio has a positive quantity of shares for that security. In contrast, if you have an open short position, your portfolio has a negative quantity of shares for that security.

Short Availability

To short sell, you need to borrow the shares from another investor or your brokerage. The short availability is the number of shares available for you to borrow. If you don't model the short availability in backtests, you could have a strategy that performs well in backtesting but doesn't trade in live mode because the shares aren't available to borrow. If you assume your short orders are successful in backtests, your backtesting results likely won't represent the true performance of actually trading the strategy during the backtest period. To get the short availability in your algorithms, use the US Equities Short Availability dataset . For more information about this topic, see Short Availability .

Borrowing Costs

In live trading, your brokerage charges an interest fee when you borrow shares to short a security. In backtesting, use the ShortMarginInterestRateModel .

To get valid borrowing rates, use the InteractiveBrokersShortableProvider .

C#

```
public override void Initialize()
{
    var security = AddEquity("SPY");
    // For US Equity, use the ShortMarginInterestRateModel with IB data
    security.SetShortableProvider(new InteractiveBrokersShortableProvider());
    security.MarginInterestRateModel = new ShortMarginInterestRateModel();
}
```

Examples

The following examples demonstrate some common practices for shorting US Equities:

Example 1: Shorting Penny Stock Top Gainers

The following algorithm adds a universe of penny stocks that have experienced the greatest upward momentum over the last two trading days. For each asset in the universe, it monitors an hourly Bollinger Bands indicator. When an asset breaks above the upper band, it shorts the asset. It exits each position when the asset drops 10% from the entry price or when the price reverts back below the middle Bollinger Band.

C#

```
using QuantConnect.Data.Shortable;
public class USEquityShortingExampleAlgorithm : QCAlgorithm
{
    private Dictionary<Symbol, RateOfChange> _rocBySymbol = [];

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Enable automatic indicator warm-up.
        Settings.AutomaticIndicatorWarmUp = true;
        // Add a universe of low-cap penny stocks that have high upward momentum.
        AddUniverse(SelectAssets);
    }

    private IEnumerable<Symbol> SelectAssets(IEnumerable<Fundamental> fundamentals)
    {
        // Select all the penny stocks that have a momentum score.
        var pennyStocks = new List<Symbol>();
        foreach (var f in fundamentals)
        {
            // Create a RateOfChange indicator for each asset.
            if (!_rocBySymbol.TryGetValue(f.Symbol, out var roc))
            {
                _rocBySymbol[f.Symbol] = roc = new RateOfChange(1);
            }
            // Update the indicator and ensure it's ready.
            // Filter out assets above $5M market cap or $1 share price.
            if (!roc.Update(f.EndTime, f.Price) || f.MarketCap > 5e6 || f.Price > 1)
            {
                continue;
            }
            pennyStocks.Add(f.Symbol);
        }

        // Select the 20 penny stocks with the greatest momentum.
        return pennyStocks.OrderByDescending(symbol => _rocBySymbol[symbol].Current.Value).Take(20);
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        // Iterate through the assets that entered the universe.
        foreach (var security in changes.AddedSecurities)
        {
            // Set the shortable provider of the asset to ensure realistic short availability.
            security.SetShortableProvider(new InteractiveBrokersShortableProvider());
            // Create a BollingerBand indicator for this asset.
            var bb = BB(security.Symbol, 20, 2, resolution: Resolution.Hour);
            // Attach an event handler to the indicator to track its updates.
            bb.Updated += OnBBUpdated;
            (security as dynamic).Bbands = bb;
        }

        // Liquidate and remove the indicator updates for assets that leave the universe.
        foreach (var security in changes.RemovedSecurities)
        {
            DeregisterIndicator((security as dynamic).Bbands);
            Liquidate(security.Symbol);
        }
    }

    private void OnBBUpdated(object indicator, IndicatorDataPoint _security_by_bb)
    {
        var bb = indicator as BollingerBands;
        // When the indicator updates, ensure it's ready.
        if (!bb.IsReady)
        {
            return;
        }
        var security = Securities[bb.Current.Symbol];
        // Short the asset if the price is above the upper band.
        if (!security.Invested && security.Price > bb.UpperBand.Current.Value)
        {
            // Get the number of shares avaiable for shorting.
            var shortableQuantity = security.ShortableProvider.ShortableQuantity(security.Symbol, Time) ?? 0m;
            // Calculate the target quantity.
            var quantity = Math.Max(CalculateOrderQuantity(security.Symbol, -0.1m), -shortableQuantity);
            if (quantity == 0) return;

            // Short the asset.
            MarketOrder(security.Symbol, quantity);
```

```
            // Place a take-profit order 10% below the current price.
            LimitOrder(security.Symbol, -quantity, 0.9m * security.BidPrice);
        }
        // If the limit order isn't hit before the price drops below the middle band,
        // exit the position and cancel the limit order.
        else if (security.Invested && security.Price < bb.MiddleBand.Current.Value)
        {
            Liquidate(security.Symbol);
        }
    }
}
```

US Equity

Data Preparation

Introduction

The US Equities dataset provides price data for backtests and live trading.

Sourcing

The QuantConnect data provider consolidates US Equity market data across all of the exchanges. Over-the-Counter (OTC) trades are excluded. The data is powered by the Securities Information Processor (SIP), so it has 100% market coverage. In contrast, free platforms that display data feeds like the Better Alternative Trading System (BATS) only have about 6-7% market coverage .

We provide live splits, dividends, and corporate actions for US companies. We deliver them to your algorithm before the trading day starts.

Bar Building

We aggregate ticks to build bars.

The bar-building process can exclude ticks. If a tick is excluded, its volume is aggregated in the bar but its price is not aggregated in the bar. Ticks are excluded if any of the following statements are true:

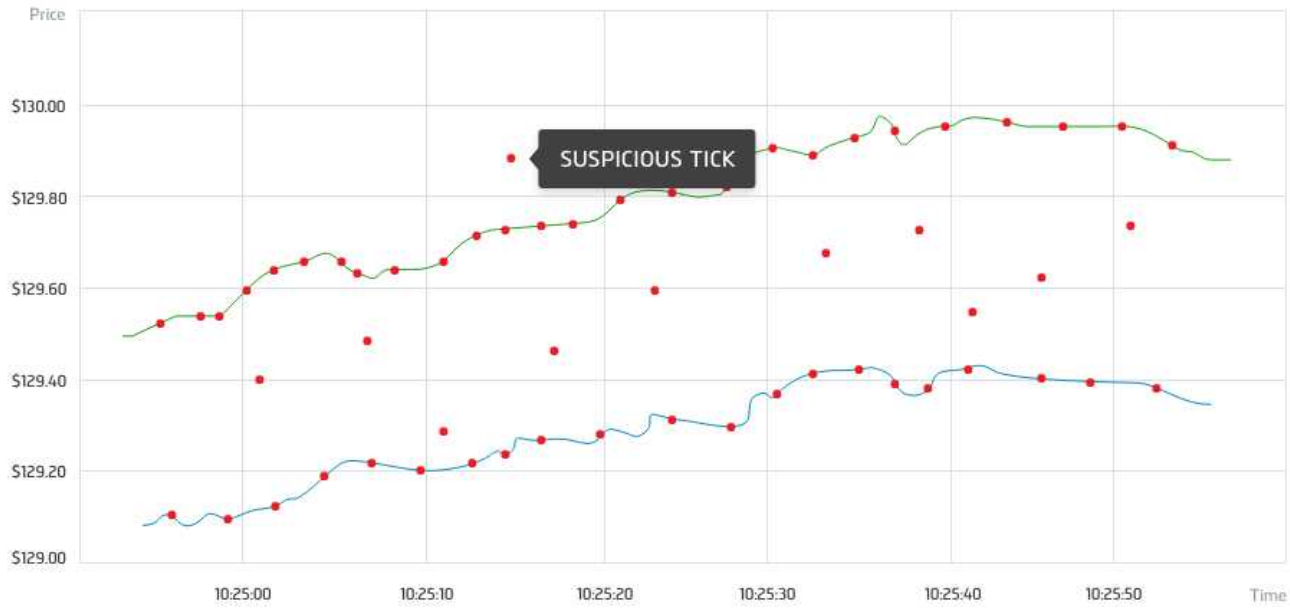
- The tick is suspicious.
- The tick is from the FINRA exchange and meets our price and volume thresholds.
- The trade has none of the following included `TradeConditionFlags` and at least one of the following excluded `TradeConditionFlags` :
  echo file_get_contents(DOCS_RESOURCES."/data-feeds/trade-condition-flags-table.html"); ?>
- The quote has a size of less than 100 shares.
- The quote has none of the following included `QuoteConditionFlags` and at least one of the following excluded `QuoteConditionFlags` :
- The quote has one of the following `QuoteConditionFlags` :
  echo file_get_contents(DOCS_RESOURCES."/data-feeds/quote-condition-flags-table.html"); ?>

In the preceding tables, Participant refers to the entities on page 19 of the Consolidated Tape System Multicast Output Binary Specification .

Suspicious Ticks

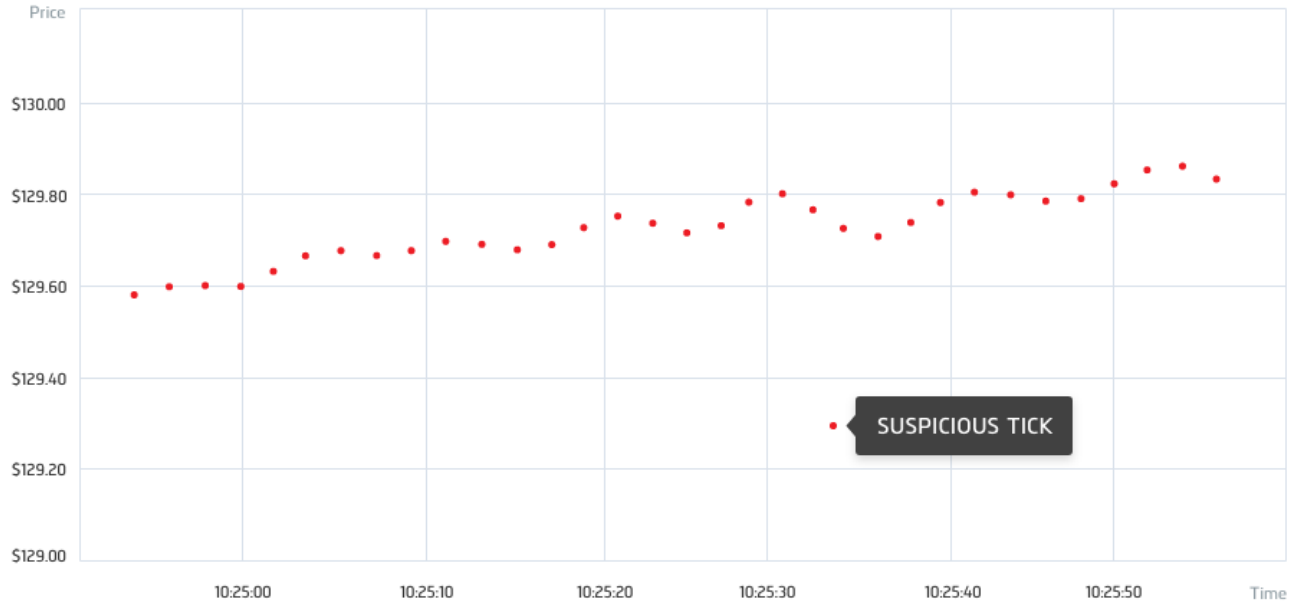
Tick price data is raw and unfiltered, so it can contain a lot of noise. If a tick is not tradable, we flag it as suspicious. This process makes the bars a more realistic representation of what you could execute in live trading. If you use tick data, avoid using suspicious ticks in your algorithms as informative data points. We recommend only using tick data if you understand the risks and are able to perform your own tick filtering. Ticks are flagged as suspicious in the following situations:

- The tick occurs below the best bid or above the best ask



This image shows a tick that occurred above the best ask price of a security. The green line represents the best ask of the security, the blue line represents the best bid of the security, and the red dots represent trade ticks. The ticks between the best bid and ask occur from filling hidden orders. The tick that occurred above the best ask price is flagged as suspicious.

- The tick occurs far from the current market price



This image shows a tick that occurred far from the price of the security. The red dots represent trade ticks. The tick that occurred far from the market price is flagged as suspicious.

- The tick occurs on a dark pool
- The tick is rolled back
- The tick is reported late

Market Auction Prices

The opening and closing price of the day is set by very specific opening and closing auction ticks. When a stock like Apple is listed, it's listed on Nasdaq. The open auction tick on Nasdaq is the price that's used as the official open of the day. NYSE,

BATS, and other exchanges also have opening auctions, but the only official opening price for Apple is the opening auction on the exchange where it was listed.

We set the opening and closing prices of the first and last bars of the day to the official auction prices. This process is used for second, minute, hour, and daily bars for the 9:30 AM and 4:00 PM Eastern Time (ET) prices. In contrast, other platforms might not be using the correct opening and closing prices.

The official auction prices are usually emitted 2-30 seconds after the market open and close. We do our best to use the official opening and closing prices in the bars we build, but the delay can be so large that there isn't enough time to update the opening and closing price of the bar before it's injected into your algorithms. For example, if you subscribe to second resolution data, we wait until the end of the second for the opening price but most second resolution data won't get the official opening price. If you subscribe to minute resolution data, we wait until the end of the minute for the opening auction price. Most of the time, you'll get the actual opening auction price with minute resolution data, but there are always exceptions. Nasdaq and NYSE can have delays in publishing the opening auction price, but we don't have control over those issues and we have to emit the data on time so that you get the bar you are expecting.

Live and Backtesting Differences

In live trading, bars are built using the exchange timestamps with microsecond accuracy. This microsecond-by-microsecond processing of the ticks can mean that the individual bars between live trading and backtesting can have slightly different ticks. As a result, it's possible for a tick to be counted in different bars between backtesting and live trading, which can lead to bars having slightly different open, high, low, close, and volume values.

Data Availability

In live trading, live data is available in real-time. In backtests, live data is available at the following pre-market trading session.

Examples

The following examples demonstrate some common practices related to our US Equity data preparation process.

Example 1: Trading at the Opening Auction

The following algorithm creates a Scheduled Event that places a Market On Open order , which fill at the official opening auction price.

C#

```
public class USEquityDataPreparationExampleAlgorithm : QCAlgorithm
{
    private Symbol _spy;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add Equity data.
        _spy = AddEquity("SPY", extendedMarketHours: true).Symbol;
        // Create a Scheduled Event that rebalances with market on open orders.
        Schedule.On(DateRules.EveryDay(_spy), TimeRules.BeforeMarketOpen(_spy, 30), Rebalance);
    }

    public void Rebalance()
    {
        var quantity = CalculateOrderQuantity(_spy, 0.1m);
        if (quantity != 0)
        {
            MarketOnOpenOrder(_spy, quantity);
        }
    }
}
```

Example 2: Demonstrate Using Tick Flags to Identify Opening Auction Tick

This example demonstrates using the tick flags to identify the official opening price on the security primary exchange. All exchanges have an opening auction to determine the market on open price, but only the primary listed exchange is used as the official open price for the asset. The algorithm subscribes to the raw tick data of QQQ ETF and logs the auction price tick data of the asset.

C#

```
public class USEquityDataPreparationExampleAlgorithm : QCAlgorithm
{
    private Equity _asset;
    uint officialOpen;
    uint openingPrints;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 5);
        SetEndDate(2024, 9, 5);
        // Subscribe to tick data of a security and find the exchange code.
        _asset = AddEquity("QQQ", Resolution.Tick, extendedMarketHours: true, dataNormalizationMode:
DataNormalizationMode.Raw);
        // Define flags to help filter opening auctions in the OnData method.
        officialOpen = (uint) (TradeConditionFlags.Regular | TradeConditionFlags.OfficialOpen);
        openingPrints = (uint) (TradeConditionFlags.Regular | TradeConditionFlags.OpeningPrints);
        // Display exchange code and trade condition flags on the console.
        Debug($"Search Codes: Ex:{_asset.PrimaryExchange.Code} OO: {officialOpen} OP: {openingPrints}");
    }

    public override void OnData(Slice slice)
    {
        // Create a list of opening auction tick data.
        var trades = _asset.Cache.GetAll<Tick>()
            .Where(x =>
                x.TickType == TickType.Trade &&
                x.Price > 0 &&
                !string.IsNullOrWhiteSpace(x.SaleCondition) && (
                    (x.ParsedSaleCondition == officialOpen || x.ParsedSaleCondition == openingPrints)
                )
                );
        // Log the opening auction tick price
        foreach(var tick in trades)
        {
            Log($"{Time},{tick.Price},{tick.Quantity},{tick.ParsedSaleCondition},{tick.ExchangeCode},
{tick.Exchange}");
        }
    }
}
```

US Equity

Market Hours

Introduction

This page shows the trading hours, holidays, and time zone of the US Equity market.

Pre-market Hours

The following table shows the pre-market hours for the US Equity market:

Weekday	**Time (America/New York)**
Monday	04:00:00 to 09:30:00
Tuesday	04:00:00 to 09:30:00
Wednesday	04:00:00 to 09:30:00
Thursday	04:00:00 to 09:30:00
Friday	04:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the US Equity market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:00:00
Tuesday	09:30:00 to 16:00:00
Wednesday	09:30:00 to 16:00:00
Thursday	09:30:00 to 16:00:00
Friday	09:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the US Equity market:

Weekday	Time (America/New York)
Monday	16:00:00 to 20:00:00
Tuesday	16:00:00 to 20:00:00
Wednesday	16:00:00 to 20:00:00
Thursday	16:00:00 to 20:00:00
Friday	16:00:00 to 20:00:00

Holidays

LEAN uses the trading holidays from the NYSE website.

The following table shows the dates of holidays for the US Equity market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28

2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04

2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the US Equity market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/New York)**
1999-11-26	13:00:00
2000-07-03	13:00:00
2000-11-24	13:00:00
2001-07-03	13:00:00
2001-11-23	13:00:00
2001-12-24	13:00:00
2002-07-05	13:00:00
2002-11-29	13:00:00
2002-12-24	13:00:00
2003-07-03	13:00:00
2003-11-28	13:00:00
2003-12-24	13:00:00
2003-12-26	13:00:00
2004-11-26	13:00:00
2005-11-25	13:00:00
2006-07-03	13:00:00

2006-11-24	13:00:00
2007-07-03	13:00:00
2007-11-23	13:00:00
2007-12-24	13:00:00
2008-07-03	13:00:00
2008-11-28	13:00:00
2008-12-24	13:00:00
2009-11-27	13:00:00
2009-12-24	13:00:00
2010-11-26	13:00:00
2011-11-25	13:00:00
2012-07-03	13:00:00
2012-11-23	13:00:00
2012-12-24	13:00:00
2013-07-03	13:00:00
2013-11-29	13:00:00
2013-12-24	13:00:00
2014-07-03	13:00:00
2014-11-28	13:00:00
2014-12-24	13:00:00
2015-11-27	13:00:00
2015-12-24	13:00:00
2016-11-25	13:00:00
2017-07-03	13:00:00
2017-11-24	13:00:00
2017-12-24	13:00:00
2018-07-03	13:00:00
2018-11-23	13:00:00

2018-12-24	13:00:00
2019-07-03	13:00:00
2019-11-29	13:00:00
2019-12-24	13:00:00
2020-11-27	13:00:00
2020-12-24	13:00:00
2021-11-26	13:00:00
2022-11-25	13:00:00
2023-07-03	13:00:00
2023-11-24	13:00:00
2024-07-03	13:00:00
2024-11-29	13:00:00
2024-12-24	13:00:00
2025-07-03	13:00:00
2025-11-28	13:00:00
2025-12-24	13:00:00
2026-11-27	13:00:00
2026-12-24	13:00:00

Late Opens

There are no days with late opens.

Time Zone

The US Equity market trades in the `America/New York` time zone.

Asset Classes

Equity Options

Equity Options are a financial derivative that gives the holder the right (but not the obligation) to buy or sell the underlying Equity, such as Apple, at the stated exercise price.

Requesting Data

Handling Data

Greeks and Implied Volatility

Market Hours

See Also

BasicTemplateOptionsAlgorithm.py
BasicTemplateOptionsAlgorithm.cs
Introduction to Options
Applied Options

Equity Options

Requesting Data

Request Equity Options data in your algorithm to receive a feed of contract prices in the `OnData` method. For more information about the specific dataset we use for backtests, see the US Equity Options dataset listing . To trade Equity Options live, you can use one of the brokerage data providers . We currently only support American-style Options for US Equity Options.

To request Equity Options data, call one of the following methods:

- `AddOption` to add a basket of contracts for a single underlying Equity
- `AddOptionContract` to add individual contracts for an underlying Equity

Request and Filter Contracts for Option Strategies

`AddOption`

Request and Filter All Contracts

`AddOption`

Select Individual Contracts

`AddOptionContract`

See Also

Introduction to Options
Applied Options

Requesting Data

Universes

Introduction

The `AddOption` method enables you to select a basket of Option contracts for an underlying Equity. To form your universe of contracts, you can filter them down by their strike price and expiry or you can choose to a subset of the contracts that form popular Option strategies . If you want to subscribe to individual contracts one-by-one instead of a set of contracts, see Individual Contracts .

Create Universes

To add a universe of Equity Option contracts, in the `Initialize` method, call the `AddOption` method. This method returns an `Option` object, which has a `SetFilter` method you can call to filter the set of tradable contract down to just the contracts you want.

```
C#
public class BasicOptionAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        UniverseSettings.Asynchronous = true;
        var option = AddOption("SPY");
        option.SetFilter(Filter);
        _symbol = option.Symbol;
    }

    private OptionFilterUniverse Filter(OptionFilterUniverse universe)
    {
        return universe.IncludeWeeklys().Expiration(0, 7).Delta(0.35m, 0.75m);
    }

    public override void OnData(Slice data)
    {
        if (data.OptionChains.TryGetValue(_symbol, out var chain))
        {
            var contract = chain.OrderBy(x => x.Greeks.Delta).FirstOrDefault();
        }
    }
}
```

For more information about the `AddOption` method, see Create Universes .

Filter by Investment Strategy

Options trading strategies consist of simultaneously buying or selling one or more Option contracts of the same underlying Equity with different features.

```C#
// Example 1: Select a Straddle
option.SetFilter(optionFilterUniverse => optionFilterUniverse.Straddle(30, 5, 10));

// Example 2: Select the contracts (including weeklys) that expire in the next 30 days that form an Iron Condor
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().Strikes(-20, 20).Expiration(0,
30).IronCondor(30, 5, 10));

// Example 3: Select the 0DTE contracts that form an Strangle
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().Expiration(0, 0).Strangle(30, 5, -
10));
```

The following table describes the filter methods of the `OptionFilterUniverse` class that select contracts for Option strategies:

`NakedCall(int minDaysTillExpiry, decimal strikeFromAtm)` Selects a call contract to form Naked Call , Covered Call , or Protective Call Option strategies.
`NakedPut(int minDaysTillExpiry, decimal strikeFromAtm)` Selects a put contract to form Naked Put , Covered Put , or Protective Put Option strategies.
`CallSpread(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects two call contracts to form Bull Call Spread or Bear Call Spread Option strategies.
`PutSpread(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects two put contracts to form Bull Put Spread or Bear Put Spread Option strategies.
`CallCalendarSpread(decimal strikeFromAtm, int minNearDaysTillExpiry, int minFarDaysTillExpiry)` Selects two call contracts to form Long Call Calendar Spread or Short Call Calendar Spread Option strategies.
`PutCalendarSpread(decimal strikeFromAtm, int minNearDaysTillExpiry, int minFarDaysTillExpiry)` Selects two put contracts to form Long Put Calendar Spread or Short Put Calendar Spread Option strategies.
`Strangle(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects two contracts to form Long Strangle or Short Strangle Option strategies.
`Straddle(int minDaysTillExpiry)` Selects two contracts to form Long Straddle or Short Straddle Option strategies.
`ProtectiveCollar(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects two contracts to form Protective Collar Option strategies.

`Conversion(int minDaysTillExpiry, decimal strikeFromAtm)` Selects two contracts to form Conversion or Reverse Conversion Option strategies.
`CallButterfly(int minDaysTillExpiry, decimal strikeSpread)` Selects three contracts to form Long Call Butterfly or Short Call Butterfly Option strategies.
`PutButterfly(int minDaysTillExpiry, decimal strikeSpread)` Selects three contracts to form Long Put Butterfly or Short Put Butterfly Option strategies.
`IronButterfly(int minDaysTillExpiry, decimal strikeSpread)` Selects four contracts to form Long Iron Butterfly or Short Iron Butterfly Option strategies.
`IronCondor(int minDaysTillExpiry, decimal nearStrikeSpread, decimal farStrikeSpread)` Selects four contracts to form Long Iron Condor or Short Iron Condor Option strategies.
`BoxSpread(int minDaysTillExpiry, decimal strikeSpread)` Selects four contracts to form Box Spread or Short Box Spread Option strategies.
`JellyRoll(decimal strikeFromAtm, int minNearDaysTillExpiry, int minFarDaysTillExpiry)` Selects four contracts to form Jelly Roll or Short Jelly Roll Option strategies.
`CallLadder(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal middleStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects four contracts to form Bear Call Ladder or Bull Call Ladder Option strategies.
`PutLadder(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal middleStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects four contracts to form Bear Put Ladder or Bull Put Ladder Option strategies.

The preceding methods return an `OptionFilterUniverse` , so you can chain the methods together.

Filter by Implied Volatility and Greeks

Option price models compute the theoretical price of Option contracts, their implied volatility, and their Greek values. Theoretical prices can help you detect undervalued and overvalued contracts, implied volatility can provide you insight into the upcoming volatility of the underlying security, and Greek values can help you hedge your portfolio.

```C#
// Example 1: Select the contracts with delta between 0.25 and 0.75
option.SetFilter(optionFilterUniverse => optionFilterUniverse.Delta(0.25m, 0.75m));

// Example 2: Select the contracts (including weeklys) that expire in the next 90 days with implied volatility below
20%
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().IV(0, 20).Expiration(0, 90));

// Example 3: Select the contracts (including weeklys) that expire in the next 30 days with implied volatility below
20% that forms an Iron Condor
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().IV(0, 20).Expiration(0,
30).IronCondor(30, 5, 10));
```

The following table describes the filter methods of the `OptionFilterUniverse` class that select contract for a range of implied volatility, Greek values, and open interest:

`IV(decimal min, decimal max)`
`ImpliedVolatility(decimal min, decimal max)`
 	
Selects a contract with implied volatility within the range you set.

`D(decimal min, decimal max)`
`Delta(decimal min, decimal max)`
 	
Selects a contract with delta within the range you set.

`G(decimal min, decimal max)`
`Gamma(decimal min, decimal max)`
 	
Selects a contract with gamma within the range you set.

`R(decimal min, decimal max)`
`Rho(decimal min, decimal max)`
 	
Selects a contract with rho within the range you set.

`V(decimal min, decimal max)`
`Vega(decimal min, decimal max)`
 	
Selects a contract with vega within the range you set.

`T(decimal min, decimal max)`
`Theta(decimal min, decimal max)`
 	
Selects a contract with theta within the range you set.

`OI(long min, long max)`
`OpenInterest(decimal min, decimal max)`
 	
Selects a contract with open interest within the range you set.

The preceding methods return an `OptionFilterUniverse` , so you can chain the methods together.

To perform thorough filtering on the `OptionFilterUniverse` , define an isolated filter method.

```C#
// Select the put contracts with the lowest strike price.
option.SetFilter(Selector);

private OptionFilterUniverse Selector(OptionFilterUniverse optionFilterUniverse)
{
    return universe
        .Delta(0.5m, 1.5m)
        .Gamma(0.0001m, 0.0006m)
        .Vega(0.01m, 1.5m)
        .Theta(-2.0m, -0.5m)
        .Rho(0.5m, 3.0m)
        .ImpliedVolatility(1.0m, 3.0m)
        .OpenInterest(100, 500);
}
```

Filter by Other Contract Properties

To set a contract filter, in the `Initialize` method, call the `SetFilter` method of the `Option` object.

```C#
// Select contracts that have a strike price within 1 strike level above and below the underlying price
option.SetFilter(minStrike: -1, maxStrike: 1);

// Select contracts that expire within 30 days
option.SetFilter(minExpiry: TimeSpan.FromDays(0), maxExpiry: TimeSpan.FromDays(30));

// Select contracts that have a strike price within 1 strike level and expire within 30 days
option.SetFilter(minStrike: -1, maxStrike: 1, minExpiry: TimeSpan.FromDays(0), maxExpiry: TimeSpan.FromDays(30));

// Select call contracts
option.SetFilter(optionFilterUniverse => optionFilterUniverse.CallsOnly());
```

The following table describes the available filter techniques:

`SetFilter(int minStrike, int maxStrike)` Selects the contracts that have a strike price within a minimum and maximum strike level relative to the underlying price. For example, say the underlying price is 302*andtherearestrikesatevery*5. If you set `minStrike` to -1 and `maxStrike` to 1, LEAN selects the contracts that have a strike of 300*or*305. This filter runs asynchronously by default.
`SetFilter(TimeSpan minExpiry, TimeSpan maxExpiry)` Selects the contracts that expire within the range you set. This filter runs asynchronously by default.
`SetFilter(int minStrike, int maxStrike, TimeSpan minExpiry, TimeSpan maxExpiry)` Selects the contracts that expire and have a strike within the range you set. This filter runs asynchronously by default.
`SetFilter(Func<OptionFilterUniverse, OptionFilterUniverse> universeFunc)` Selects the contracts that a function selects.

The following table describes the filter methods of the `OptionFilterUniverse` class:

`Strikes(int minStrike, int maxStrike)` Selects contracts that are within `minStrike` strikes below the underlying price and `maxStrike` strikes above the underlying price.
`CallsOnly()` Selects call contracts.
`PutsOnly()` Selects put contracts.
`StandardsOnly()` Selects standard contracts.
`IncludeWeeklys()` Selects non-standard weeklys contracts.
`WeeklysOnly()` Selects weekly contracts.
`FrontMonth()` Selects the front month contract.
`BackMonths()` Selects the non-front month contracts.
`BackMonth()` Selects the back month contracts.
`Expiration(int minExpiryDays, int maxExpiryDays)` Selects contracts that expire within a range of dates relative to the current day.
`Contracts(IEnumerable<Symbol> contracts)` Selects a list of contracts.
`Contracts(Func<IEnumerable<Symbol>, IEnumerable< Symbol>> contractSelector)` Selects contracts that a selector function selects.

The preceding methods return an `OptionFilterUniverse`, so you can chain the methods together.

```
// Example 1: Select the front month call contracts
option.SetFilter(optionFilterUniverse => optionFilterUniverse.CallsOnly().FrontMonth());

// Example 2: Select the contracts (including weeklys) that expire in the next 90 days
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().Strikes(-20, 20).Expiration(0, 90));

// Example 3: Select the contracts (including weeklys) that expire in the next 30 days that form an Iron Condor
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().Strikes(-20, 20).Expiration(0,
30).IronCondor(30, 5, 10));
```

Some of the preceding filter methods only set an internal enumeration in the `OptionFilterUniverse` that it uses later on in the filter process. This subset of filter methods don't immediately reduce the number of contract `Symbol` objects in the `OptionFilterUniverse`.

Default Filter

By default, LEAN subscribes to the Option contracts that have the following characteristics:

- Standard type (exclude weeklys)
- Within 1 strike price of the underlying asset price
- Expire within 35 days

To adjust the universe of contracts, set a filter. The filter usually runs at the first bar of every day. When the filter selects a contract that isn't currently in your universe, LEAN adds the new contract data to the next `Slice` that it passes to the `OnData` method.

Historical Data

To get historical chains for an Equity Option, see Historical Data.

Examples

Example 1: Selecting for 0DTE Contracts

0DTE Options are Option contracts that expire on the same day you trade them. To create an universe with 0DTE Options, call the `SetFilter` method with expiration set. In order to trade with the contracts with most liquidity, limit the strike range from 10 strikes above and below the current price level.

```
public class EquityOptionAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        var option = AddOption("SPY");
        // Filter for 0-DTE contracts within +/-10 strike range.
        // They have the highest volatility but low premium, providing sufficient liquidity and higher profit margin.
        option.SetFilter(u => u.IncludeWeeklys().Expiration(0, 0).Strikes(-10, 10));
    }
}
```

Example 2: Straddle

You can also create an universe that return the option contracts that form an option strategy. In this example, we demonstrate filtering for a long straddle strategy that expires 30 days later.

```C#
public class EquityOptionAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        var option = AddOption("SPY");
        _symbol = option.Symbol;
        // Using Staddle() method, it will only return the best-matched ATM call and put contracts expiring after 30
days.
        // It provides better accuracy in filtering, and subscribe only to the needed contracts to save computation
resources.
        option.SetFilter(u => u.IncludeWeeklys().Straddle(30));
    }

    public override void OnData(Slice slice)
    {
        // Only wants the option chain of the selected symbol.
        if (!Portfolio.Invested &&
            slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            if (chain.Count == 0)
            {
                return;
            }
            // There should only be 1 expiry and 1 strike from the 2 contracts returned, getting from either contract
is fine.
            var expiry = chain.First().Expiry;
            var strike = chain.First().Strike;

            // Forms a long straddle option strategy using abstraction method ensure accuracy.
            var longStraddle = OptionStrategies.Straddle(_symbol, strike, expiry);
            Buy(longStraddle, 1);
        }
    }
}
```

Example 3: Custom Selector

Using a custom selection function, you can filter the option universe to obtain the contracts that you need without subscribing the redundants. This can ensure the accuracy of the filter, as well as avoiding unnecessary subscriptions that drag the algorithm computational performance. In this example, we try to obtain a put contract that expires 30 days later that is 5*belowthecurrentunderlyingpriceandacallcontractthatexpires*60*dayslaterthatis*10 above the current underlying price.

C#

```
public class EquityOptionAlgorithm : QCAlgorithm
{
    private Equity _underlying;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Seed the price with last known price to ensure the underlying price data is available on initial option
contract filtering.
        Settings.SeedInitialPrices = true;
        // Subscribe to underlying data for ATM calculation using the update underlying price.
        // Set data normalization mode to raw is required to ensure strike price and underlying price is comparable.
        _underlying = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw);
        // Subscribe to SPY option data.
        var option = AddOption(_underlying.Symbol);
        option.SetFilter(u => u.IncludeWeeklys().Contracts((contracts) =>
        {
            var selected = new List<Symbol>();

            // Select the best-match put contracts that expires 30 days later that is $5 below the current underlying
price.
            // We should first filter the hard requirement, then sort the reminders for faster computation.
            var puts = contracts.Where(x =>
                    x.ID.OptionRight == OptionRight.Put &&
                    x.ID.Date >= Time.AddDays(30) &&
                    x.ID.StrikePrice <= _underlying.Price - 5m
                ).ToList();
            if (puts.Count > 0)
            {
                var put = puts.OrderBy(x => x.ID.Date)
                    .ThenByDescending(x => x.ID.StrikePrice)
                    .First();
                selected.Add(put);
            }

            // Select the best-match call contract that expires 60 days later that is $10 above the current
underlying price.
            var calls = contracts.Where(x =>
                    x.ID.OptionRight == OptionRight.Call &&
                    x.ID.Date >= Time.AddDays(60) &&
                    x.ID.StrikePrice >= _underlying.Price + 10m
                ).ToList();
            if (calls.Count > 0)
            {
                var call = calls.OrderBy(x => x.ID.Date)
                    .ThenBy(x => x.ID.StrikePrice)
                    .First();
                selected.Add(call);
            }

            return selected;
        }));
    }
}
```

Requesting Data

Individual Contracts

Introduction

The `AddOptionContract` method enables you to add an individual Option contract to your algorithm. To check which contracts are currently available to add to your algorithm, use the `OptionChain` method. If you want to subscribe to a set of contracts instead of individual contracts one-by-one, see Universes .

Create Subscriptions

Before you can subscribe to an Option contract, you must configure the underlying Equity and get the contract `Symbol` .

```
C#
public class BasicOptionAlgorithm : QCAlgorithm
{
    private Symbol _underlying, _contractSymbol;
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        _underlying = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;
    }

    public override void OnData(Slice data)
    {
        if (_contractSymbol != null)
        {
            return;
        }
        var chain = OptionChain(_underlying);
        var expiry = chain.Select(contract => contract.Expiry).Min();
        _contractSymbol = chain
            .Where(contract =>
                contract.Expiry == expiry &&
                contract.Right == OptionRight.Call &&
                contract.Greeks.Delta > 0.3m &&
                contract.Greeks.Delta < 0.7m
            )
            .OrderByDescending(contract => contract.OpenInterest)
            .First()
            .Symbol;
        AddOptionContract(_contractSymbol);
    }
}
```

Configure the Underlying Equity

If you want to subscribe to the underlying Equity in the `Initialize` method, set the Equity data normalization to `DataNormalizationMode.Raw` .

```
C#
_underlying = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;
```

If your algorithm has a dynamic universe of Equities, before you add the Equity universe in the `Initialize` method, set the universe data normalization mode to `DataNormalizationMode.Raw` .

```
UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;
```

Get Contract Symbols

To subscribe to an Option contract, you need the contract `Symbol`. The preferred method to getting Option contract `Symbol` objects is to use the `OptionChain` method. This method returns an `OptionChain` object, which represent an entire chain of Option contracts for a single underlying security. With the chain, sort and filter the data to find the specific contract(s) you want to trade.

```
// Get the contracts available to trade.
var chain = OptionChain(_underlying);

// Select a contract.
var expiry = chain.Select(contract => contract.Expiry).Min();
_contractSymbol = chain
    .Where(contract =>
        // Select call contracts with the closest expiry.
        contract.Expiry == expiry &&
        contract.Right == OptionRight.Call &&
        // Select contracts with a 0.3-0.7 delta.
        contract.Greeks.Delta > 0.3m &&
        contract.Greeks.Delta < 0.7m
    )
    // Select the contract with the largest open interest.
    .OrderByDescending(contract => contract.OpenInterest)
    .First()
    // Get the Symbol of the target contract.
    .Symbol;
```

Alternatively, you can create a Option contract `Symbol` object directly parsing its OSI ticker using the `SymbolRepresentation.ParseOptionTickerOSI` method.

```
// Create the contract Symbol for the SPY Call 564 3 Sept 2024.
_contractSymbol = SymbolRepresentation.ParseOptionTickerOSI("SPY 240903C00564000");
```

Subscribe to Contracts

To create an Equity Option contract subscription, pass the contract `Symbol` to the `AddOptionContract` method. Save a reference to the contract `Symbol` so you can easily access the Option contract in the OptionChain that LEAN passes to the `OnData` method. This method returns an `Option` object. To override the default pricing model of the Option, set a pricing model .

```
var option = AddOptionContract(_contractSymbol);
option.PriceModel = OptionPriceModels.BinomialCoxRossRubinstein();
```

The `AddOptionContract` method creates a subscription for a single Option contract and adds it to your user-defined universe. To create a dynamic universe of Option contracts, add an Equity Options universe or an Options Universe Selection model .

Warm Up Contract Prices

If you subscribe to an Option contract with `AddOptionContract` , you'll need to wait until the next `Slice` to receive data and trade the contract. To trade the contract in the same time step you subscribe to the contract, set the current price of the contract in a security initializer .

```
Settings.SeedInitialPrices = true;
```

Supported Assets

To view the supported assets in the US Equities dataset, see the Data Explorer .

Resolutions

The following table shows the available resolutions and data formats for Equity Option contract subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`				
`Second`				
`Minute`	✓	✓		
`Hour`	✓	✓		
`Daily`	✓	✓		

The default resolution for Option contract subscriptions is `Resolution.Minute` . To change the resolution, pass a `resolution` argument to the `AddOptionContract` method.

```
AddOptionContract(_contractSymbol, Resolution.Minute);
```

To create custom resolution periods, see Consolidating Data .

Supported Markets

LEAN groups all of the US Equity exchanges under `Market.USA` . You don't need to pass a `Market` argument to the `AddOptionContract` method because the contract `Symbol` already contains the market.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
AddOptionContract(_contractSymbol, fillForward: false);
```

Margin and Leverage

LEAN models buying power and margin calls to ensure your algorithm stays within the margin requirements. Options are already leveraged products, so you can't change their leverage.

Extended Market Hours

By default, your security subscriptions only cover regular trading hours. To subscribe to pre and post-market trading hours for a

specific asset, enable the `extendedMarketHours` argument when you create the security subscription.

```
C#
AddOptionContract(_contractSymbol, extendedMarketHours: true);
```

You only receive extended market hours data if you create the subscription with minute, second, or tick resolution. If you create the subscription with daily or hourly resolution, the bars only reflect the regular trading hours.

To view the schedule of regular and extended market hours, see Market Hours .

Data Normalization

The data normalization mode doesn't affect the data that LEAN passes to `OnData` or the data from history request . By default, LEAN doesn't adjust Equity Options data for splits and dividends of their underlying. If you change the data normalization mode, it won't change the outcome.

If you hold an Option contract when a corporate action occurs for the underlying Equity, LEAN automatically closes your position.

Remove Subscriptions

To remove a contract subscription that you created with `AddOptionContract` , call the `RemoveOptionContract` method. This method is an alias for `RemoveSecurity` .

```
C#
RemoveOptionContract(_contractSymbol);
```

The `RemoveOptionContract` method cancels your open orders for the contract and liquidates your holdings.

Properties

The `AddOptionContract` method returns an `Option` object, which have the following properties:

Helper Methods

The `Option` object provides methods you can use for basic calculations. These methods require the underlying price. To get the `Option` object and the `Security` object for its underlying in any function, use the Option `Symbol` to access the value in the `Securities` object.

```
C#
var option = Securities[_contractSymbol];
var underlying = Securities[_contractSymbol.Underlying];
var underlyingPrice = underlying.Price;
```

To get the Option payoff , call the `GetPayOff` method.

```
C#
var payOff = option.GetPayOff(underlyingPrice);
```

To get the Option intrinsic value , call the `GetIntrinsicValue` method.

```
C#
var intrinsicValue = option.GetIntrinsicValue(underlyingPrice);
```

To get the Option out-of-the-money amount , call the `OutOfTheMoneyAmount` method.

```
var otmAmount = option.OutOfTheMoneyAmount(underlyingPrice);
```

To check whether the Option can be automatic exercised, call the `IsAutoExercised` method.

```
var isAutoExercised = option.IsAutoExercised(underlyingPrice);
```

Exceptions and Edge Cases

The following sections explain exceptions and edge cases with subscribing to individual Option contracts.

Manually Creating Option Symbol Objects

To subscribe to an Option contract, you need the contract `Symbol` . You can get the contract `Symbol` from the `CreateOption` or `OptionChain` methods. If you use the `CreateOption` method, you need to provide the details of an existing contract.

```
_contractSymbol = QuantConnect.Symbol.CreateOption(_underlying, Market.USA,
    OptionStyle.American, OptionRight.Call, 365, new DateTime(2022, 6, 17));
```

Default Underlying Subscription Settings

If you subscribe to an Equity Option contract but don't have a subscription to the underlying Equity, LEAN automatically subscribes to the underlying Equity with the following settings:

Setting	Value
Fill forward	Same as the Option contract
Leverage	0
Extended Market Hours	Same as the Option contract
Data Normalization	`DataNormalizationMode.Raw`

In this case, you still need the Equity `Symbol` to subscribe to Equity Option contracts. If you don't have access to it, create it.

```
_underlying = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
```

Overriding the Initial Implied Volatility Guess

To override the initial guess of implied volatility , set and warm up the underlying volatility model .

Examples

The following examples demonstrate some common practices for requesting individual Equity Option contract data.

Example 1: Covered Call

A cover call consists of a short call and with a lot of the underlying equity. Although it capped the maximum profit if the

underlying skyrocketted, it also provide extra credit received while speculating the underlying will rise.

```C#
public class EquityOptionExampleAlgorithm : QCAlgorithm
{
    private Symbol _aapl;
    private List<Symbol> _options = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Seed the security price to ensure the retrieval of the ATM calls at the initial filtering.
        Settings.SeedInitialPrices = true;
        // Set the data normalization mode as raw for option strike-price comparability.
        _aapl = AddEquity("AAPL", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;

        // Filter an updated option list at market open everyday by a scheduled event.
        Schedule.On(
            DateRules.EveryDay(_aapl),
            TimeRules.AfterMarketOpen(_aapl, 0),
            RefreshOptionList
        );
    }

    private void RefreshOptionList()
    {
        // Select the calls expires within 30 days and within $5 strike from ATM as leg of the covered call.
        // $5 buffer is given on selecting the ATM call due to price movement.
        _options = OptionChain(_aapl)
            .Where(x =>
                x.Expiry < Time.AddDays(30) &&
                x.Right == OptionRight.Call &&
                Math.Abs(x.Strike - Securities[_aapl].Price) <= 5
            )
            .Select(symbol => AddOptionContract(symbol).Symbol)
            .ToList();
    }

    public override void OnData(Slice slice)
    {
        if (!Portfolio.Invested && slice.Bars.ContainsKey(_aapl) && _options.Count > 0)
        {
            // To form a covered call, get the contract closest to ATM and expiry.
            var contract = _options.OrderBy(x => x.ID.Date)
                .ThenBy(x => Math.Abs(x.ID.StrikePrice - slice.Bars[_aapl].Price))
                .First();

            // Covered call involves shorting 1 ATM call and ordering 1 lot of underlying.
            MarketOrder(contract, -1);
            MarketOrder(_aapl, Securities[contract].SymbolProperties.ContractMultiplier);
        }
    }

    public override void OnOrderEvent(OrderEvent orderEvent)
    {
        // Close AAPL position if the short call is not exercised (OTM).
        // If it is exercised, the underlying will be used to settle the contract automatically.
        if (orderEvent.Ticket.OrderType == OrderType.OptionExercise &&
            !orderEvent.IsInTheMoney)
        {
            MarketOrder(_aapl, -Securities[orderEvent.Symbol].SymbolProperties.ContractMultiplier);
        }
    }
}
```

Example 2: 0-DTE Bull Put Spread

0DTE options often trades with high volume and volatility, providing arbitration opportunities and higher profit margin on spread type trading. In this example, we make use of 0-DTE SPY options to trade bull put spread option strategy.

C#

```
public class EquityOptionExampleAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private List<Leg> _legs = [];

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Seed the underlying security price to ensure accurate filtering for puts of $5 above/below current market
price.
        Settings.SeedInitialPrices = true;
        // Set the data normalization mode as raw for option strike-price comparability.
        _spy = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;

        // Filter an updated option list at market open everyday by a scheduled event.
        Schedule.On(DateRules.EveryDay(_spy), TimeRules.BeforeMarketOpen(_spy, 10), RefreshOptionList);

        // Use a scheduled event to close all positions before market close.
        Schedule.On(DateRules.EveryDay(_spy), TimeRules.BeforeMarketClose(_spy, 1), ExitPositions);
    }

    private void RefreshOptionList()
    {
        // Get all tradable option contracts for SPY at the current time for filtering.
        var filteredContracts = OptionChain(_spy)
            // Select the 0-DTE puts by setting expiry within 1 day.
            .Where(x => x.Expiry < Time.AddDays(1) && x.Right == OptionRight.Put)
            .ToList();

        // Ensure at least 2 contracts available to form a put spread.
        if (filteredContracts.Count < 2)
        {
            _legs = [];
            return;
        }
        // To form a put spread, select and subscribe to put contracts $5 above and below from the current underlying
price.
        var underlyingPrice = Securities[_spy].Price;
        var itmPut = filteredContracts.MinBy(x => Math.Abs(x.Strike - underlyingPrice - 5));
        var otmPut = filteredContracts.MinBy(x => Math.Abs(x.Strike - underlyingPrice + 5));
        Leg getLeg(OptionContract contract, int quantity)
        {
            AddOptionContract(contract.Symbol);
            return Leg.Create(contract.Symbol, quantity);
        }
        _legs = [ getLeg(itmPut, -1), getLeg(otmPut, 1) ];
    }

    public override void OnData(Slice slice)
    {
        // To avoid over-trading, limit the position opening to before 3pm.
        // To ensure the put spread formed correctly, make sure at least 2 contracts selected.
        if (!Portfolio.Invested && Time.Hour < 15 && _legs.Count == 2)
        {
            // A bull put spread involves buying a lower-strike put and selling a high-strike put
            _legs.ForEach(leg => MarketOrder(leg.Symbol, leg.Quantity));
        }
    }

    private void ExitPositions()
    {
        // Exit all positions by removing the contract before market close to avoid option assignment and overnight
risk.
        _legs.ForEach(leg => RemoveOptionContract(leg.Symbol, "Exit position"));
        _legs.Clear();
    }
}
```

Example 3: Select Option Contracts by Greeks

The following algorithm creates a weekly Scheduled Event that adds a universe of contracts with a Delta >= 0.99. This universe is useful for creating a hedge replicate portfolio for arbitrage.

C#

```
public class EquityOptionExampleAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private List<Option> _universe = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add the underlying asset with raw data to enable Option strike-price comparability.
        _underlying = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;
        // Add a Scheduled Event to select the Option contracts to trade each week.
        Schedule.On(
            DateRules.WeekStart(_underlying),
            TimeRules.At(7, 30),
            AddContracts
        );
    }

    private void AddContracts()
    {
        _universe.Clear();
        // Get all the tradable Option contracts for SPY at the current time.
        var chain = OptionChain(_underlying);
        // Add the contracts that have the nearest expiry after 7 days and a delta >= 0.99.
        var expiryThreshold = Time.AddDays(7);
        var contracts = chain.Where(contract => contract.Expiry >= expiryThreshold);
        if (!contracts.IsNullOrEmpty())
        {
            var expiry = contracts.Select(contract => contract.Expiry).Min();
            _universe = contracts
                .Where(contract => contract.Expiry == expiry & contract.Greeks.Delta >= 0.99m)
                .Select(contract => AddOptionContract(contract.Symbol))
                .ToList();
        }
        // Plot the number of contracts in the universe.
        Plot("Universe", "Size", _universe.Count);
    }
}
```

Example 4: Wheel Strategy

The Wheel strategy is a popular trading strategy for Options that enables traders to build a steady flow of income from Equity assets they want to hold for the long term.

C#

```
public class EquityOptionExampleAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    // Set OTM threshold for wheel strategy profit margin.
    private decimal _otmThreshold = 0.05m;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 5);
        SetEndDate(2024, 9, 5);
        SetCash(1000000000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Set the data normalization mode as raw for option strike-price comparability.
        _spy = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;
    }

    public override void OnData(Slice slice)
    {
        // To use the latest underlying price to filter the option contract, ensure the SPY in the bar data.
        // Open short put contract position only when the last wheel is completed.
        if (!Portfolio.Invested && slice.Bars.ContainsKey(_spy))
        {
            // Initiate the wheel by shorting a least-OTM put contract that the strike is below the threshold.
            var symbol = GetTargetContract(OptionRight.Put, slice.Bars[_spy].Price * (1 - _otmThreshold));
            SetHoldings(symbol, -0.2m);
        }
        // Open short call contract position only when the put is assigned (portfolio with the underlying) to close
the wheel and underlying position by the call assignment.
        else if (Portfolio[_spy].Invested && slice.Bars.ContainsKey(_spy))
        {
            // Short the corresponding number of a least-OTM call contract that the strike is above the threshold.
            var symbol = GetTargetContract(OptionRight.Call, slice.Bars[_spy].Price * (1 + _otmThreshold));
            MarketOrder(symbol, -Portfolio[_spy].Quantity / Securities[_spy].SymbolProperties.ContractMultiplier);
        }
    }

    private Symbol GetTargetContract(OptionRight right, decimal targetPrice)
    {
        // Get all tradable option contracts for SPY at the current time for filtering.
        var contractSymbols = OptionChain(_spy).ToList();
        // Filter for the least-OTM contract that is off by the threshold to form the wheel.
        // Expiry is set to be at least 30 days to earn the time decay, which is highest in the last month.
        var expiry = contractSymbols.Where(x => x.ID.Date > Time.AddDays(30))
            .Min(x => x.ID.Date);
        var filtered = contractSymbols.Where(x =>
                x.ID.Date == expiry &&
                x.ID.OptionRight == right &&
                (right == OptionRight.Call ? x.ID.StrikePrice >= targetPrice : x.ID.StrikePrice <= targetPrice)
            )
            .OrderBy(x => x.ID.StrikePrice)
            .ToList();
        var selected = right == OptionRight.Call ? filtered.First() : filtered.Last();
        // Request the selected contract data for trading.
        return AddOptionContract(selected).Symbol;
    }
}
```

For more details, refer to the Wheel Strategy research post .

Example 5: Scan and Update Option Chain Every 5 Minutes

The following example shows how to update the Option chain every five minutes. The `OptionChainManager` class implements the selection logic and manages the contract subscriptions.

C#

```
public class OptionChainFullExample : QCAlgorithm
{
    private Dictionary<Symbol, OptionChainManager> _chainManager = new();
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(100000);
```

```
        UniverseSettings.Asynchronous = true;
        UniverseSettings.MinimumTimeInUniverse = TimeSpan.Zero;
        // Seed the security price to ensure the underlying price data is ready at the initial filtering
        Settings.SeedInitialPrices = true;
        // Set the data normalization mode as raw for option strike-price comparability
        var spy = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;
        // Set up a OptionChainManager to filter the contracts based on latest data by request
        _chainManager[QuantConnect.Symbol.CreateCanonicalOption(spy)] = new(-10, 10, 0, 7);
        // Daily population of the available contracts to ensure the contracts are tradable
        PopulateOptionChain();
        Schedule.On(DateRules.EveryDay(spy), TimeRules.AfterMarketOpen(spy, 1), PopulateOptionChain);
        // Filter for contract in every 5 minutes interval through scheduled event
        Schedule.On(DateRules.EveryDay(spy), TimeRules.Every(TimeSpan.FromMinutes(5)), Filter);
    }

    private void PopulateOptionChain()
    {
        // The contract list is updated daily, so we can get it and apply
        // the expiration filter as soon as the market opens.
        foreach (var (symbol, manager) in _chainManager)
        {
            manager.SetChain(OptionChain(symbol), Time);
        }
        Filter();
    }

    private void Filter()
    {
        foreach (var (symbol, manager) in _chainManager)
        {
            manager.Select(this, symbol);
        }
    }

    public override void OnData(Slice slice)
    {
        // Iterate the saved symbol and chain manager to obtain only the contract wanted
        foreach (var (symbol, manager) in _chainManager)
        {
            if (!slice.OptionChains.TryGetValue(symbol, out var chain))
                continue;
            // Handle option exercise and assignment on unwanted underlying position
            if (Portfolio[symbol.Underlying].Invested)
            {
                Liquidate(symbol.Underlying);
            }

            // Buy the ATM call with nearest expiry to speculate-trade the underlying through low cost, leveraging
position
            var expiry = chain.Min(x => x.Expiry);
            var atmCall = chain
                .Where(x => x.Expiry == expiry && x.Right == OptionRight.Call && Securities[x.Symbol].IsTradable)
                .OrderBy(x => Math.Abs(chain.Underlying.Price - x.Strike))
                .FirstOrDefault();
            if (atmCall != null && !Portfolio[atmCall.Symbol].Invested)
            {
                MarketOrder(atmCall.Symbol, 1);
            }
        }
    }
}

internal class OptionChainManager
{
    private readonly int _minStrike;
    private readonly int _maxStrike;
    private readonly int _minExpiry;
    private readonly int _maxExpiry;
    private List<OptionContract> _chain = new();
    private readonly List<Symbol> _symbols = new();

    public OptionChainManager(int minStrike, int maxStrike, int minExpiry, int maxExpiry)
    {
        _minStrike = minStrike;
        _maxStrike = maxStrike;
        _minExpiry = minExpiry;
        _maxExpiry = maxExpiry;
    }

    public void SetChain(OptionChain chain, DateTime time)
```

```
    {
        // Expiry criteria will not affect intra-day universe filtering, so it is done in a daily basis in higher
level
        _chain = chain.Where(x =>
        {
            var totalDays = (x.Expiry - time).TotalDays;
            return _minExpiry <= totalDays && totalDays <= _maxExpiry;
        }).ToList();
    }

    public void Select(QCAlgorithm algorithm, Symbol underlyingSymbol)
    {
        if (_chain.IsNullOrEmpty())
            return;
        if (underlyingSymbol.IsCanonical())
            underlyingSymbol = underlyingSymbol.Underlying;

        var strikes = _chain.Select(x => x.Strike).OrderBy(x => x).Distinct().ToList();
        var spot = algorithm.Securities[underlyingSymbol].Price;
        var atm = strikes.OrderBy(x => Math.Abs(spot - x)).FirstOrDefault();
        var index = strikes.IndexOf(atm);
        var minStrike = strikes[Math.Max(0, index + _minStrike)];
        var maxStrike = strikes[Math.Min(strikes.Count - 1, index + _maxStrike)];
        var symbols = _chain
            .Where(x => minStrike <= x.Strike && x.Strike <= maxStrike)
            .Select(x => x.Symbol).ToList();

        // Also remove data subscription on the contracts being filtered out to release computation resources
            foreach (var symbol in _symbols.Except(symbols).ToList())
        {
            if (algorithm.RemoveOptionContract(symbol))
                _symbols.Remove(symbol);
        }
        foreach (var symbol in symbols.Except(_symbols).ToList())
        {
            _symbols.Add(symbol);
            algorithm.AddOptionContract(symbol);
        }
    }
}
```

Equity Options

Handling Data

Introduction

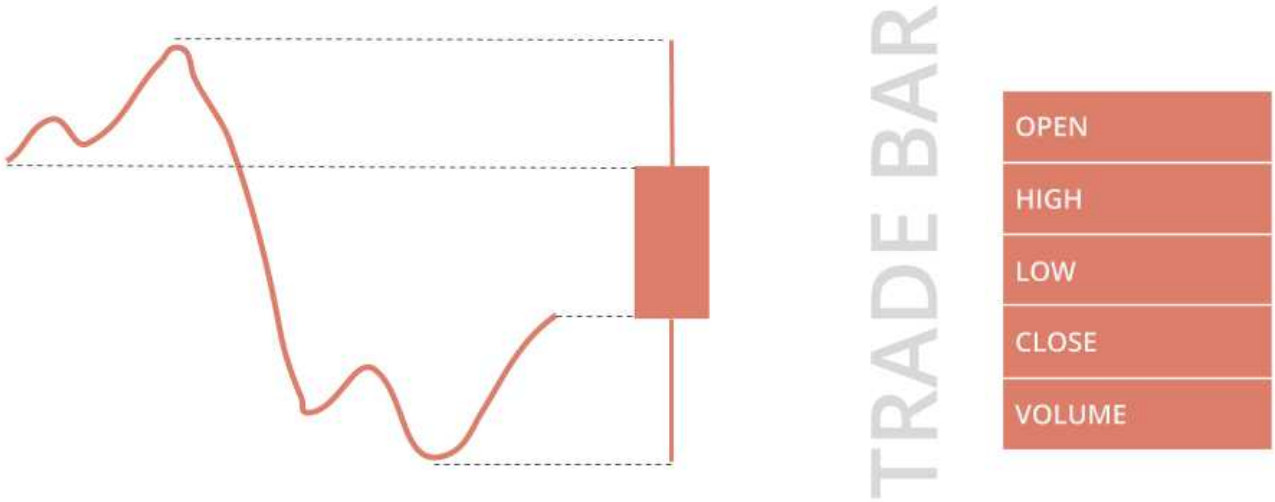
LEAN passes the data you request to the `OnData` method so you can make trading decisions. The default `OnData` method accepts a `Slice` object, but you can define additional `OnData` methods that accept different data types. For example, if you define an `OnData` method that accepts a `TradeBar` argument, it only receives `TradeBar` objects. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `TradeBars` `DataDictionary` is made up of `TradeBar` objects. To access individual data points in the dictionary, you can index the dictionary with the contract ticker or `Symbol` , but we recommend you use the `Symbol` .

To view the resolutions that are available for Equity Options data, see Resolutions .

Trades

`TradeBar` objects are price bars that consolidate individual trades from the exchanges. They contain the open, high, low, close, and volume of trading activity over a period of time.



To get the `TradeBar` objects in the `Slice` , index the `Slice` or index the `Bars` property of the `Slice` with the contract `Symbol` . If the contract doesn't actively trade or you are in the same time step as when you added the contract subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your contract before you index the `Slice` with the contract `Symbol` .

```C#
public override void OnData(Slice slice)
{
    // Check if the symbol is contained in TradeBars object
    if (slice.Bars.ContainsKey(_contractSymbol))
    {
        // Obtain the mapped TradeBar of the symbol
        var tradeBar = slice.Bars[_contractSymbol];
    }
}
```

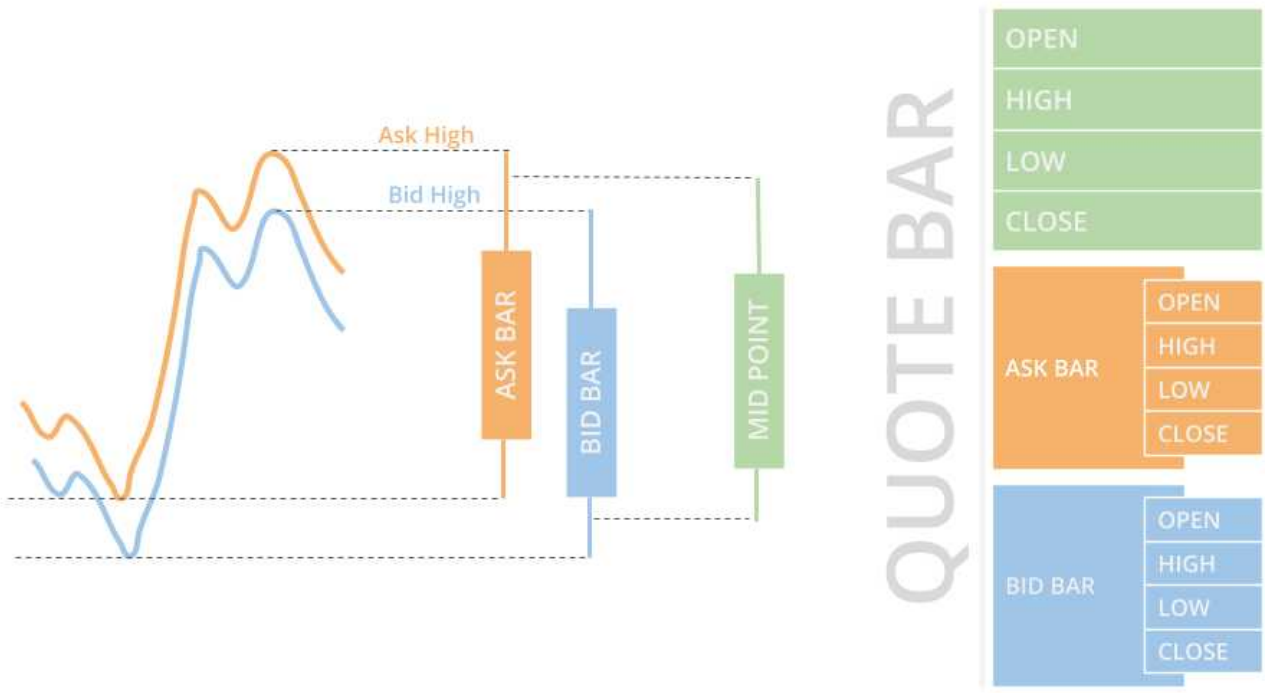
You can also iterate through the `TradeBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `TradeBar` objects.

```C#
public override void OnData(Slice slice)
{
    // Iterate all received Symbol-TradeBar key-value pairs
    foreach (var kvp in slice.Bars)
    {
        var symbol = kvp.Key;
        var tradeBar = kvp.Value;
        var closePrice = tradeBar.Close;
    }
}
```

`TradeBar` objects have the following properties:

Quotes

`QuoteBar` objects are bars that consolidate NBBO quotes from the exchanges. They contain the open, high, low, and close prices of the bid and ask. The `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` object are the mean of the respective bid and ask prices. If the bid or ask portion of the `QuoteBar` has no data, the `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` copy the values of either the `Bid` or `Ask` instead of taking their mean.



To get the `QuoteBar` objects in the `Slice` , index the `QuoteBars` property of the `Slice` with the contract `Symbol` . If the contract doesn't actively get quotes or you are in the same time step as when you added the contract subscription, the `Slice` may not

contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your contract before you index the `Slice` with the contract `Symbol` .

```
C#
public override void OnData(Slice slice)
{
    // Check if the symbol is contained in QuoteBars object
    if (slice.QuoteBars.ContainsKey(_contractSymbol))
    {
        // Obtain the mapped QuoteBar of the symbol
        var quoteBar = slice.QuoteBars[_contractSymbol];
    }
}
```

You can also iterate through the `QuoteBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `QuoteBar` objects.

```
C#
public override void OnData(Slice slice)
{
    // Iterate all received Symbol-QuoteBar key-value pairs
    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        var askPrice = quoteBar.Ask.Close;
    }
}
```

`QuoteBar` objects let LEAN incorporate spread costs into your simulated trade fills to make backtest results more realistic.

`QuoteBar` objects have the following properties:

Option Chains

`OptionChain` objects represent an entire chain of Option contracts for a single underlying security.

To get the `OptionChain` , index the `OptionChains` property of the `Slice` with the canonical `Symbol` .

```
C#
public override void OnData(Slice slice)
{
    // Try to get the OptionChain using the canonical symbol
    if (slice.OptionChains.TryGetValue(_contractSymbol.Canonical, out var chain))
    {
        // Get all contracts if the OptionChain contains any member
        var contracts = chain.Contracts;
    }
}
```

You can also loop through the `OptionChains` property to get each `OptionChain` .

```
C#
public override void OnData(Slice slice)
{
    // Iterate all received Canonical Symbol-OptionChain key-value pairs
    foreach (var kvp in slice.OptionChains)
    {
        var canonicalSymbol = kvp.Key;
        var chain = kvp.Value;
        var contracts = chain.Contracts;
    }
}
```

`OptionChain` objects have the following properties:

Option Contracts

`OptionContract` objects represent the data of a single Option contract in the market. To get the Option contracts in the `Slice` , use the `Contracts` property of the `OptionChain` .

```
C#
public override void OnData(Slice slice)
{
    // Try to get the OptionChain using the canonical symbol
    if (slice.OptionChains.TryGetValue(_contractSymbol.Canonical, out var chain))
    {
        // Get individual contract data
        if (chain.Contracts.TryGetValue(_contractSymbol, out var contract))
        {
            var price = contract.Price;
        }
    }
}
```

Greeks and Implied Volatility

To get the Greeks and implied volatility of an Option contract, use the `Greeks` and `implied_volatility` members.

```
C#
public override void OnData(Slice slice)
{
    // Try to get the OptionChain using the canonical symbol
    if (slice.OptionChains.TryGetValue(_contractSymbol.Canonical, out var chain))
    {
        // Get individual contract data
        if (chain.Contracts.TryGetValue(_contractSymbol, out var contract))
        {
            // Get greeks data of the selected contract
            var delta = contract.Greeks.Delta;
            var iv = contract.ImpliedVolatility;
        }
    }
}
```

LEAN only calculates Greeks and implied volatility when you request them because they are expensive operations. If you invoke the `Greeks` property, the Greeks aren't calculated. However, if you invoke the `Greeks.Delta` , LEAN calculates the delta. To avoid unecessary computation in your algorithm, only request the Greeks and implied volatility when you need them. For more information about the Greeks and implied volatility, see Options Pricing .

Open Interest

Open interest is the number of outstanding contracts that haven't been settled. It provides a measure of investor interest and the market liquidity, so it's a popular metric to use for contract selection. Open interest is calculated once per day. To get the latest open interest value, use the `OpenInterest` property of the `Option` or `OptionContract` .

```C#
public override void OnData(Slice slice)
{
    // Try to get the OptionChains using the canonical symbol
    if (slice.OptionChains.TryGetValue(_contractSymbol.Canonical, out var chain))
    {
        // Get individual contract data
        if (chain.Contracts.TryGetValue(_contractSymbol, out var contract))
        {
            // Get the open interest of the selected contracts
            var openInterest = contract.OpenInterest;
        }
    }
}

public void OnData(OptionChains optionChains)
{
    // Try to get the OptionChains using the canonical symbol
    if (optionChains.TryGetValue(_contractSymbol.Canonical, out var chain))
    {
        // Get individual contract data
        if (chain.Contracts.TryGetValue(_contractSymbol, out var contract))
        {
            // Get the open interest of the selected contracts
            var openInterest = contract.OpenInterest;
        }
    }
}
```

Properties

`OptionContract` objects have the following properties:

Examples

Example 1: Get Mid Price For Individual Contracts

This example shows how to handle `QuoteBar` data for shortlisted Equity Option contracts to calculate mid price using bid close and ask close data, while filter individual option contracts and request data using `OptionChain` method for the contracts that expires within the current week. Using mid price, we can examine the market fair value of the Option and compare with model theoretical price.

C#

```
public class EquityOptionHandlingDataAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private List<Symbol> _contracts = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);

        // Seed the price with last known price to ensure the underlying price data is available on initial option
contract filtering.
        Settings.SeedInitialPrices = true;
        // Subscribe to underlying data for ATM calculation using the update underlying price.
        // Set data normalization mode to raw is required to ensure strike price and underlying price is comparable.
        _spy = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;

        // Update the tradable contracts daily before market open since the option contract list provider populate
them daily.
        Schedule.On(
            DateRules.EveryDay(_spy),
            TimeRules.At(9, 0),
            UpdateContracts
        );
    }

    private void UpdateContracts()
    {
        // Get all contracts that expiring this week to trade with, subscribe to data for trading need.
        _contracts = OptionChain(_spy)
            .Where(x => x.ID.Date <= Expiry.EndOfWeek(Time))
            .Select(x => AddOptionContract(x).Symbol)
            .ToList();
    }

    public override void OnData(Slice slice)
    {
        // Only focus on filtered list of option contracts to trade.
        foreach (var contract in _contracts)
        {
            // Mid price can only be calculated when quote bar data is available.
            if (slice.QuoteBars.TryGetValue(contract, out var quote))
            {
                if (quote.Bid != null && quote.Ask != null)
                {
                    // Mid price = average of bid close price and ask close price.
                    var midPrice = (quote.Bid.Close + quote.Ask.Close) * 0.5m;
                }
            }
        }
    }
}
```

Example 2: Get Mid Price For Universe

This example shows how to handle `QuoteBar` data for shortlisted Equity Option contracts to calculate mid price using bid close and ask close data, while request data through universe selection function using `SetFilter` method for the contracts that expires within the current week. Using mid price, we can examine the market fair value of the Option and compare with model theoretical price.

```C#
public class EquityOptionHandlingDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);

        // Seed the price with last known price to ensure the underlying price data is available on initial option
contract filtering.
        Settings.SeedInitialPrices = true;
        // Subscribe to underlying data for ATM calculation using the update underlying price.
        // Set data normalization mode to raw is required to ensure strike price and underlying price is comparable.
        var spy = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;
        // Subscribe to SPY option data.
        var option = AddOption(spy);
        _symbol = option.Symbol;
        // We wish to only trade the contracts expiring within the same week since they have the highest volume.
        option.SetFilter((u) => u.IncludeWeeklys().Contracts((x) => x.Where(s => s.ID.Date <=
Expiry.EndOfWeek(Time))));
    }

    public override void OnData(Slice slice)
    {
        // Only want to obtain the option chain of the selected symbol.
        if (slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            foreach (var contract in chain)
            {
                // Mid price = average of bid close price and ask close price.
                var midPrice = (contract.BidPrice + contract.AskPrice) * 0.5m;
            }
        }
    }
}
```

Example 3: Get Instant Delta

The option greeks change rapidly, so we need to obtain the instant greeks to accurately calculate the hedge size for arbitration or replication portfolio. You can call the `Greeks` property from the `OptionChain` data to access various greeks. In this example, we will demonstrate how to obtain the contract with delta closest to 0.4 among all call contracts expiring the same week.

C#

```
public class EquityOptionHandlingDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 5);
        SetEndDate(2024, 9, 5);

        // Seed the price with last known price to ensure the underlying price data is available on initial option
contract filtering.
        Settings.SeedInitialPrices = true;
        // Subscribe to underlying data for ATM calculation using the update underlying price.
        // Set data normalization mode to raw is required to ensure strike price and underlying price is comparable.
        var spy = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;
        // Subscribe to SPY option data.
        var option = AddOption(spy);
        _symbol = option.Symbol;
        // We wish to only trade the contracts expiring within the same week since they have the highest volume.
        // 0.4 Delta will only appear in call contracts.
        option.SetFilter((u) => u.IncludeWeeklys().Delta(0.2m, 0.6m).Contracts((x) => x.Where(s => s.ID.Date <=
Expiry.EndOfWeek(Time))));
    }

    public override void OnData(Slice slice)
    {
        // Only want to obtain the option chain of the selected symbol.
        if (slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            // Get the contract with Delta closest to 0.4 to trade.
            // The arbitary delta criterion might be set due to hedging need or risk adjustment.
            var selected = chain.OrderBy(x => Math.Abs(x.Greeks.Delta - 0.4m)).First();
        }
    }
}
```

Equity Options

Greeks and Implied Volatility

The Greeks are measures that describe the sensitivity of an Option's price to various factors like underlying price changes (Delta), time decay (Theta), volatility (Vega), and interest rates (Rho), while Implied Volatility (IV) represents the market's expectation of the underlying asset's volatility over the life of the Option.

Key Concepts

Concepts on Greeks, IV, and IV modeling

Indicators

Implementation of IV and Greeks as LEAN indicators

See Also

Option Pricing
Risk Free Interest Rate
Dividend Yield

Greeks and Implied Volatility

Key Concepts

Introduction

LEAN provides multiple ways for you to get the Greeks and implied volatility (IV) of Option contracts. There is a daily, pre-calculated dataset based on the end of the previous trading day. There is also a stream of values that are calculated on-the-fly as your algorithm runs.

Greeks on Universes

The Greeks and IV values in the filter function are the daily, pre-calculated values based on the end of the previous trading day.

```C#
public override void Initialize()
{
    var option = AddOption("SPY");
    option.SetFilter(universe => universe.IncludeWeeklys().Delta(0.3m, 0.7m).Expiration(0, 7));
}
```

To calculate the values, we use our implementation of the forward tree pricing model, which accounts for the interest rate and dividend payments.

You can't customize the Greeks and IV values that the filter function receives. However, you can create indicators to customize how the Greeks and IV are calculated for the contracts already in your universe.

Greek History

The Greeks and IV values that you get from a history request of the Option universe are the daily, pre-calculated values based on the end of the previous trading day.

```C#
public class OptionHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        var option = AddOption("SPY");
        var history = History<OptionUniverse>(option.Symbol, 5);
        foreach (var chain in history)
        {
            var endTime = chain.EndTime;
            var filteredContracts = chain.Data
                .Select(contract => contract as OptionUniverse)
                .Where(contract => contract.Greeks.Delta > 0.3m);
            foreach (var contract in filteredContracts)
            {
                var price = contract.Price;
                var iv = contract.ImpliedVolatility;
            }
        }
    }
}
```

You can't customize the Greeks and IV values that you get from a history request. However, you can create indicators to customize how the Greeks and IV are calculated for the contracts already in your universe.

Greeks on Data Events

The Greeks and IV values that you get from the current Slice are from the price model calculations. These values aren't the same as the values you get in the universe filter function.

```
C#
public class OptionDataEventsAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        var option = AddOption("GOOG");
        option.PriceModel = OptionPriceModels.CrankNicolsonFD();
    }

    public override void OnData(Slice data)
    {
        foreach (var chain in data.OptionChains.Values)
        {
            foreach (var contract in chain)
            {
                var iv = contract.ImpliedVolatility;
                var delta = contract.Greeks.Delta;
            }
        }
    }
}
```

To view all the models LEAN supports, see Supported Models .

Greek Indicators

The Greeks and IV values that you get from the indicators are calculated in the same way as the values you get in the filter function, but they can reflect intraday prices instead of just the daily closing prices.

For Equity Options, the `IndicatorHistory` uses the `ScaledRaw` data normalization mode when requesting historical prices.

Examples

For detailed examples on using Option Greeks and Implied Volatility indicators in algorithm, you may refer to the Greeks Indicators page examples.

Greeks and Implied Volatility

Indicators

Introduction

Option indicators enable you to calculate daily and intraday values for the Greeks and implied volatility of Option contracts. This page explains the most popular Option indicators and how to use them with LEAN.

Parameters

The following table describes the arguments that the automatic Option indicators methods accept:

Argument	Data Type	Description	Default Value
`symbol`	`Symbol`	The contract to use when calculating the indicator values.	
`mirrorOption`	`Symbol`	The mirror contract to use in parity type calculations.	`null`
`riskFreeRate`	`decimal`	The risk-free interest rate. If you don't provide a value, the default value is the US primary credit rate from the Interest Rate Provider Model .	`null`
`dividendYield`	`decimal`	The dividend yield rate. If you don't provide a value, the default value comes from the Dividend Yield Provider Model .	`null`
`optionModel`	`OptionPricingModelType`	The Option pricing model that's used to calculate the Greeks. If you don't provide a value, the default value is `OptionPricingModelType.BinomialCoxRossRubinstein` for Equity Options.	`null`
`resolution`	`Resolution`	The resolution of the indicator data. If you don't provide a value, the default value is the resolution of the subscription you have for the Option contract(s).	`null`

To perform implied volatility (IV) smoothing with a put-call pair, pass one of the contracts as the `symbol` argument and pass the other contract in the pair as the `mirrorOption` argument. The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, call the `SetSmoothingFunction` method of the `ImpliedVolatility` class/property.

Several different Option pricing models are supported to calculate the IV and Greeks. The following table describes the

`OptionPricingModelType` enumeration members:

Implied Volatility

Implied volatility, σ , is the market's expectation for the future volatility of an asset and is implied by the price of the assets's Options contracts. You can't observe it in the market but you can derive it from the price of an Option. For more information about implied volatility, see Implied Volatility .

Automatic Indicators

To create an automatic indicator for implied volatility, call the `QCAlgorithm.IV` method with the Option contract `Symbol` object(s).

C#

```
public class AutomaticImpliedVolatilityIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;

        // Set up a Scheduled Event to select contracts and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count() > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddOptionContract(contracts[0]) as dynamic;
                var contract2 = AddOptionContract(contracts[1]) as dynamic;

                // Create and save the automatic ImpliedVolatility indicators.
                contract1.IV = IV(contracts[0], contracts[1]);
                contract2.IV = IV(contracts[1], contracts[0]);

                _options = (contracts[0], contracts[1]);
            }
        }
    }
```

```
    public override void OnData(Slice slice)
    {
        // Get the ImpliedVolatility indicator of each contract.
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var symbol in chain)
            {
                var option = Securities[symbol] as dynamic;
                var indicator = option.IV;
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

For more information about the `IV` method, see Parameters and Using IV Indicator .

Manual Indicators

To create a manual indicator for implied volatility, call the `ImpliedVolatility` constructor.

C#

```
public class ManualImpliedVolatilityIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private DividendYieldProvider _dividendYieldProvider;
    // Define the Option pricing model.
    private readonly OptionPricingModelType _optionPricingModel = OptionPricingModelType.ForwardTree;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;

        // Set up dividend yield provider for the underlying
        _dividendYieldProvider = new(_underlying);

        // Set up a Scheduled Event to select contract and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
```

```
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddOptionContract(contracts[0]);
                var contract2 = AddOptionContract(contracts[1]);

                // Create and save the manual ImpliedVolatility indicators.
                foreach (var (contractA, contractB) in new[] { (contract1, contract2), (contract2, contract1) })
                {
                    var option = contractA as dynamic;
                    option.ImpliedVolatility = new ImpliedVolatility(
                            contractA.Symbol, RiskFreeInterestRateModel, _dividendYieldProvider,
                            contractB.Symbol, _optionPricingModel
                        );
                }

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var option in chain)
            {
                var indicator = (Securities[option] as dynamic).ImpliedVolatility;
                var underlying = option.Symbol.Underlying;
                var mirror = QuantConnect.Symbol.CreateOption(
                    underlying.Value, option.ID.Market, option.ID.OptionStyle,
                    option.ID.OptionRight == OptionRight.Call ? OptionRight.Put : OptionRight.Call,
                    option.ID.StrikePrice, option.ID.Date
                );

                // Check if price data is available for both contracts and the underlying asset.
                var q = slice.QuoteBars;
                var b = slice.Bars;
                if (q.ContainsKey(option) && q.ContainsKey(mirror) && b.ContainsKey(underlying))
                {
                    var dataPoints = new List<IndicatorDataPoint>
                    {
                        new IndicatorDataPoint(option, q[option].EndTime, q[option].Close),
                        new IndicatorDataPoint(mirror, q[mirror].EndTime, q[mirror].Close),
                        new IndicatorDataPoint(
                            underlying, b[underlying].EndTime, b[underlying].Close
                        )
                    };
                    foreach (var dataPoint in dataPoints)
                    {
                        indicator.Update(dataPoint);
                    }

                    // Get the current value of the ImpliedVolatility indicator.
                    var value = indicator.Current.Value;
                }
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
```

```
            Liquidate(_underlying);
        }
    }
}
```

For more information about the `ImpliedVolatility` constructor, see Using IV Indicator .

Volatility Smoothing

The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, pass a `mirrorOption` argument to the `IV` method or `ImpliedVolatility` constructor and then call the `SetSmoothingFunction` method of the resulting `ImpliedVolatility` object. The follow table describes the arguments of the custom function:

Argument	Data Type	Description
`iv`	`decimal`	The IV of the Option contract.
`mirrorIv`	`decimal`	The IV of the mirror Option contract.

The method must return a `decimal` as the smoothened IV.

C#

```
private ImpliedVolatility _iv;

public override void Initialize()
{
    var option = QuantConnect.Symbol.CreateOption("AAPL", Market.USA, OptionStyle.American, OptionRight.Put, 505m,
new DateTime(2014, 6, 27));
    AddOptionContract(option);

    var mirrorOption = QuantConnect.Symbol.CreateOption("AAPL", Market.USA, OptionStyle.American, OptionRight.Call,
505m, new DateTime(2014, 6, 27));
    AddOptionContract(mirrorOption);

    _iv = IV(option, mirrorOption);
    // example: take average of the call-put pair
    _iv.SetSmoothingFunction((iv, mirrorIv) => (iv + mirrorIv) * 0.5m);
}
```

Delta

Delta, Δ , is the rate of change of the Option price with respect to the price of the underlying asset. It measures the first-order sensitivity of the price to a movement in underlying price. For example, an Option delta of 0.4 means that if the underlying asset moves by 1%, then the value of the Option moves by $0.4 \times 1\% = 0.4\%$. For more information about delta, see Delta .

Automatic Indicators

To create an automatic indicator for delta, call the `QCAlgorithm.D` method with the Option contract `Symbol` object(s).

C#

```
public class AutomaticDeltaIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset
```

```
        // Subscribe to the underlying asset.
        _underlying = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;

        // Set up a Scheduled Event to select contracts and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count() > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddOptionContract(contracts[0]) as dynamic;
                var contract2 = AddOptionContract(contracts[1]) as dynamic;

                // Create and save the automatic Delta indicators.
                contract1.D = D(contracts[0], contracts[1]);
                contract2.D = D(contracts[1], contracts[0]);

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        // Get the Delta indicator of each contract.
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var symbol in chain)
            {
                var option = Securities[symbol] as dynamic;
                var indicator = option.D;
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

The follow table describes the arguments that the `D` method accepts in addition to the standard parameters :

Argument	Data Type	Description	Default Value
ivModel	OptionPricingModelType	The Option pricing model to use to estimate the IV when calculating Delta. If you don't provide a value, the default value is to match the optionModel parameter.	null

For more information about the D method, see Using D Indicator .

Manual Indicators

To create a manual indicator for delta, call the Delta constructor.

C#

```
public class ManualDeltaIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private DividendYieldProvider _dividendYieldProvider;
    // Define the Option pricing model.
    private readonly OptionPricingModelType _optionPricingModel = OptionPricingModelType.ForwardTree;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;

        // Set up dividend yield provider for the underlying
        _dividendYieldProvider = new(_underlying);

        // Set up a Scheduled Event to select contract and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count > 1)
            {
                // Subscribe to both contracts.
```

```
                var contract1 = AddOptionContract(contracts[0]);
                var contract2 = AddOptionContract(contracts[1]);

                // Create and save the manual Delta indicators.
                foreach (var (contractA, contractB) in new[] { (contract1, contract2), (contract2, contract1) })
                {
                    var option = contractA as dynamic;
                    option.Delta = new Delta(
                            contractA.Symbol, RiskFreeInterestRateModel, _dividendYieldProvider,
                            contractB.Symbol, _optionPricingModel
                        );
                }

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var option in chain)
            {
                var indicator = (Securities[option] as dynamic).Delta;
                var underlying = option.Symbol.Underlying;
                var mirror = QuantConnect.Symbol.CreateOption(
                    underlying.Value, option.ID.Market, option.ID.OptionStyle,
                    option.ID.OptionRight == OptionRight.Call ? OptionRight.Put : OptionRight.Call,
                    option.ID.StrikePrice, option.ID.Date
                );

                // Check if price data is available for both contracts and the underlying asset.
                var q = slice.QuoteBars;
                var b = slice.Bars;
                if (q.ContainsKey(option) && q.ContainsKey(mirror) && b.ContainsKey(underlying))
                {
                    var dataPoints = new List<IndicatorDataPoint>
                    {
                        new IndicatorDataPoint(option, q[option].EndTime, q[option].Close),
                        new IndicatorDataPoint(mirror, q[mirror].EndTime, q[mirror].Close),
                        new IndicatorDataPoint(
                            underlying, b[underlying].EndTime, b[underlying].Close
                        )
                    };
                    foreach (var dataPoint in dataPoints)
                    {
                        indicator.Update(dataPoint);
                    }

                    // Get the current value of the Delta indicator.
                    var value = indicator.Current.Value;
                }
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

For more information about the `Delta` constructor, see Using D Indicator .

Volatility Smoothing

The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, pass a `mirrorOption` argument to the indicator method or constructor and then call the

`SetSmoothingFunction` method of the `ImpliedVolatility` property of the indicator.

```
// Example: Average IV of the call-put pair.
deltaIndicator.ImpliedVolatility.SetSmoothingFunction((iv, mirrorIv) => (iv + mirrorIv) * 0.5m);
```

Gamma

Gamma, Γ, is the rate of change of the portfolio's delta with respect to the underlying asset's price. It represents the second-order sensitivity of the Option to a movement in the underlying asset's price. For more information about Gamma, see Gamma .

Automatic Indicators

To create an automatic indicator for gamma, call the `QCAlgorithm.G` method with the Option contract `Symbol` object(s).

```
public class AutomaticGammaIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;

        // Set up a Scheduled Event to select contracts and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count() > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddOptionContract(contracts[0]) as dynamic;
                var contract2 = AddOptionContract(contracts[1]) as dynamic;

                // Create and save the automatic Gamma indicators.
                contract1.G = G(contracts[0], contracts[1]);
                contract2.G = G(contracts[1], contracts[0]);
```

```
                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        // Get the Gamma indicator of each contract.
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var symbol in chain)
            {
                var option = Securities[symbol] as dynamic;
                var indicator = option.G;
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

The follow table describes the arguments that the `G` method accepts in addition to the standard parameters :

Argument	Data Type	Description	Default Value
`ivModel`	`OptionPricingModelType`	The Option pricing model to use to estimate the IV when calculating Gamma. If you don't provide a value, the default value is to match the `optionModel` parameter.	`null`

For more information about the `G` method, see Using G Indicator .

Manual Indicators

To create a manual indicator for gamma, call the `Gamma` constructor.

C#

```
public class ManualGammaIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private DividendYieldProvider _dividendYieldProvider;
    // Define the Option pricing model.
    private readonly OptionPricingModelType _optionPricingModel = OptionPricingModelType.ForwardTree;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;

        // Set up dividend yield provider for the underlying
        _dividendYieldProvider = new(_underlying);

        // Set up a Scheduled Event to select contract and create the indicators every day before market open
```

```
        // Set up a scheduled event to select contract and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddOptionContract(contracts[0]);
                var contract2 = AddOptionContract(contracts[1]);

                // Create and save the manual Gamma indicators.
                foreach (var (contractA, contractB) in new[] { (contract1, contract2), (contract2, contract1) })
                {
                    var option = contractA as dynamic;
                    option.Gamma = new Gamma(
                            contractA.Symbol, RiskFreeInterestRateModel, _dividendYieldProvider,
                            contractB.Symbol, _optionPricingModel
                        );
                }

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var option in chain)
            {
                var indicator = (Securities[option] as dynamic).Gamma;
                var underlying = option.Symbol.Underlying;
                var mirror = QuantConnect.Symbol.CreateOption(
                    underlying.Value, option.ID.Market, option.ID.OptionStyle,
                    option.ID.OptionRight == OptionRight.Call ? OptionRight.Put : OptionRight.Call,
                    option.ID.StrikePrice, option.ID.Date
                );

                // Check if price data is available for both contracts and the underlying asset.
                var q = slice.QuoteBars;
                var b = slice.Bars;
                if (q.ContainsKey(option) && q.ContainsKey(mirror) && b.ContainsKey(underlying))
                {
                    var dataPoints = new List<IndicatorDataPoint>
                    {
                        new IndicatorDataPoint(option, q[option].EndTime, q[option].Close),
                        new IndicatorDataPoint(mirror, q[mirror].EndTime, q[mirror].Close),
                        new IndicatorDataPoint(
                            underlying, b[underlying].EndTime, b[underlying].Close
                        )
                    };
```

```
                    };
                    foreach (var dataPoint in dataPoints)
                    {
                        indicator.Update(dataPoint);
                    }

                    // Get the current value of the Gamma indicator.
                    var value = indicator.Current.Value;
                }
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

For more information about the `Gamma` constructor, see Using G Indicator .

Volatility Smoothing

The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, pass a `mirrorOption` argument to the indicator method or constructor and then call the `SetSmoothingFunction` method of the `ImpliedVolatility` property of the indicator.

C#

```
// Example: Average IV of the call-put pair.
gammaIndicator.ImpliedVolatility.SetSmoothingFunction((iv, mirrorIv) => (iv + mirrorIv) * 0.5m);
```

Vega

Vega, ν , is the rate of change in the value of the Option with respect to the volatility of the underlying asset. For more information about vega, see Vega .

Automatic Indicators

To create an automatic indicator for vega, call the `QCAlgorithm.V` method with the Option contract `Symbol` object(s).

C#

```
public class AutomaticVegaIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;

        // Set up a Scheduled Event to select contracts and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }
```

```
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count() > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddOptionContract(contracts[0]) as dynamic;
                var contract2 = AddOptionContract(contracts[1]) as dynamic;

                // Create and save the automatic Vega indicators.
                contract1.V = V(contracts[0], contracts[1]);
                contract2.V = V(contracts[1], contracts[0]);

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        // Get the Vega indicator of each contract.
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var symbol in chain)
            {
                var option = Securities[symbol] as dynamic;
                var indicator = option.V;
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

The follow table describes the arguments that the `V` method accepts in addition to the standard parameters :

Argument	Data Type	Description	Default Value
ivModel	OptionPricingModelType	The Option pricing model to use to estimate the IV when calculating Vega. If you don't provide a value, the default value is to match the optionModel parameter.	null

For more information about the V method, see Using V Indicator .

Manual Indicators

To create a manual indicator for vega, call the Vega constructor.

C#

```
public class ManualVegaIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private DividendYieldProvider _dividendYieldProvider;
    // Define the Option pricing model.
    private readonly OptionPricingModelType _optionPricingModel = OptionPricingModelType.ForwardTree;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;

        // Set up dividend yield provider for the underlying
        _dividendYieldProvider = new(_underlying);

        // Set up a Scheduled Event to select contract and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count > 1)
            {
                // Subscribe to both contracts.
```

```
                var contract1 = AddOptionContract(contracts[0]);
                var contract2 = AddOptionContract(contracts[1]);

                // Create and save the manual Vega indicators.
                foreach (var (contractA, contractB) in new[] { (contract1, contract2), (contract2, contract1) })
                {
                    var option = contractA as dynamic;
                    option.Vega = new Vega(
                            contractA.Symbol, RiskFreeInterestRateModel, _dividendYieldProvider,
                            contractB.Symbol, _optionPricingModel
                        );
                }

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var option in chain)
            {
                var indicator = (Securities[option] as dynamic).Vega;
                var underlying = option.Symbol.Underlying;
                var mirror = QuantConnect.Symbol.CreateOption(
                    underlying.Value, option.ID.Market, option.ID.OptionStyle,
                    option.ID.OptionRight == OptionRight.Call ? OptionRight.Put : OptionRight.Call,
                    option.ID.StrikePrice, option.ID.Date
                );

                // Check if price data is available for both contracts and the underlying asset.
                var q = slice.QuoteBars;
                var b = slice.Bars;
                if (q.ContainsKey(option) && q.ContainsKey(mirror) && b.ContainsKey(underlying))
                {
                    var dataPoints = new List<IndicatorDataPoint>
                    {
                        new IndicatorDataPoint(option, q[option].EndTime, q[option].Close),
                        new IndicatorDataPoint(mirror, q[mirror].EndTime, q[mirror].Close),
                        new IndicatorDataPoint(
                            underlying, b[underlying].EndTime, b[underlying].Close
                        )
                    };
                    foreach (var dataPoint in dataPoints)
                    {
                        indicator.Update(dataPoint);
                    }

                    // Get the current value of the Vega indicator.
                    var value = indicator.Current.Value;
                }
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

For more information about the `Vega` constructor, see Using V Indicator .

Volatility Smoothing

The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, pass a `mirrorOption` argument to the indicator method or constructor and then call the

`SetSmoothingFunction` method of the `ImpliedVolatility` property of the indicator.

```
// Example: Average IV of the call-put pair.
vegaIndicator.ImpliedVolatility.SetSmoothingFunction((iv, mirrorIv) => (iv + mirrorIv) * 0.5m);
```

Theta

Theta, Θ, is the rate of change of the value of the Option with respect to the passage of time. It is also known to as the time decay of an Option. For more information about theta, see Theta .

Automatic Indicators

To create an automatic indicator for theta, call the `QCAlgorithm.T` method with the Option contract `Symbol` object(s).

```
public class AutomaticThetaIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;

        // Set up a Scheduled Event to select contracts and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count() > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddOptionContract(contracts[0]) as dynamic;
                var contract2 = AddOptionContract(contracts[1]) as dynamic;

                // Create and save the automatic Theta indicators.
                contract1.T = T(contracts[0], contracts[1]);
                contract2.T = T(contracts[1], contracts[0]);
```

```
                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        // Get the Theta indicator of each contract.
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var symbol in chain)
            {
                var option = Securities[symbol] as dynamic;
                var indicator = option.T;
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

The follow table describes the arguments that the `T` method accepts in addition to the standard parameters :

Argument	Data Type	Description	Default Value
`ivModel`	`OptionPricingModelType`	The Option pricing model to use to estimate the IV when calculating theta. If you don't provide a value, the default value is to match the `optionModel` parameter.	`null`

For more information about the `T` method, see Using T Indicator .

Manual Indicators

To create a manual indicator for theta, call the `Theta` constructor.

C#

```
public class ManualThetaIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private DividendYieldProvider _dividendYieldProvider;
    // Define the Option pricing model.
    private readonly OptionPricingModelType _optionPricingModel = OptionPricingModelType.ForwardTree;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;

        // Set up dividend yield provider for the underlying
        _dividendYieldProvider = new(_underlying);

        // Set up a Scheduled Event to select contract and create the indicators every day before market open
```

```
        // Set up a scheduled event to select contract and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddOptionContract(contracts[0]);
                var contract2 = AddOptionContract(contracts[1]);

                // Create and save the manual Theta indicators.
                foreach (var (contractA, contractB) in new[] { (contract1, contract2), (contract2, contract1) })
                {
                    var option = contractA as dynamic;
                    option.Theta = new Theta(
                            contractA.Symbol, RiskFreeInterestRateModel, _dividendYieldProvider,
                            contractB.Symbol, _optionPricingModel
                        );
                }

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var option in chain)
            {
                var indicator = (Securities[option] as dynamic).Theta;
                var underlying = option.Symbol.Underlying;
                var mirror = QuantConnect.Symbol.CreateOption(
                    underlying.Value, option.ID.Market, option.ID.OptionStyle,
                    option.ID.OptionRight == OptionRight.Call ? OptionRight.Put : OptionRight.Call,
                    option.ID.StrikePrice, option.ID.Date
                );

                // Check if price data is available for both contracts and the underlying asset.
                var q = slice.QuoteBars;
                var b = slice.Bars;
                if (q.ContainsKey(option) && q.ContainsKey(mirror) && b.ContainsKey(underlying))
                {
                    var dataPoints = new List<IndicatorDataPoint>
                    {
                        new IndicatorDataPoint(option, q[option].EndTime, q[option].Close),
                        new IndicatorDataPoint(mirror, q[mirror].EndTime, q[mirror].Close),
                        new IndicatorDataPoint(
                            underlying, b[underlying].EndTime, b[underlying].Close
                        )
                    };
```

```
                    };
                    foreach (var dataPoint in dataPoints)
                    {
                        indicator.Update(dataPoint);
                    }

                    // Get the current value of the Theta indicator.
                    var value = indicator.Current.Value;
                }
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

For more information about the `Theta` constructor, see Using T Indicator .

Volatility Smoothing

The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, pass a `mirrorOption` argument to the indicator method or constructor and then call the `SetSmoothingFunction` method of the `ImpliedVolatility` property of the indicator.

C#

```
// Example: Average IV of the call-put pair.
thetaIndicator.ImpliedVolatility.SetSmoothingFunction((iv, mirrorIv) => (iv + mirrorIv) * 0.5m);
```

Rho

Rho, ρ , is the rate of change of the value of a derivative with respect to the interest rate . It is usually small and not a big issue in practice unless the Option is deep in-the-money and has a long horizon. In this case, the interest rate matters because you need to discount a larger cash flow over a longer horizon. For more information about rho, see Rho .

Automatic Indicators

To create an automatic indicator for rho, call the `QCAlgorithm.R` method with the Option contract `Symbol` object(s).

C#

```
public class AutomaticRhoIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;

        // Set up a Scheduled Event to select contracts and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
```

```
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count() > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddOptionContract(contracts[0]) as dynamic;
                var contract2 = AddOptionContract(contracts[1]) as dynamic;

                // Create and save the automatic Rho indicators.
                contract1.R = R(contracts[0], contracts[1]);
                contract2.R = R(contracts[1], contracts[0]);

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        // Get the Rho indicator of each contract.
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var symbol in chain)
            {
                var option = Securities[symbol] as dynamic;
                var indicator = option.R;
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

The follow table describes the arguments that the `R` method accepts in addition to the standard parameters :

Argument	Data Type	Description	Default Value
ivModel	OptionPricingModelType	The Option pricing model to use to estimate the IV when calculating rho If you don't provide a value, the default value is to match the optionModel parameter.	null

For more information about the R method, see Using R Indicator .

Manual Indicators

To create a manual indicator for rho, call the Rho constructor.

C#

```
public class ManualRhoIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private DividendYieldProvider _dividendYieldProvider;
    // Define the Option pricing model.
    private readonly OptionPricingModelType _optionPricingModel = OptionPricingModelType.ForwardTree;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;

        // Set up dividend yield provider for the underlying
        _dividendYieldProvider = new(_underlying);

        // Set up a Scheduled Event to select contract and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddOptionContract(contracts[0]);
```

```
                var contract2 = AddOptionContract(contracts[1]);

                // Create and save the manual Rho indicators.
                foreach (var (contractA, contractB) in new[] { (contract1, contract2), (contract2, contract1) })
                {
                    var option = contractA as dynamic;
                    option.Rho = new Rho(
                            contractA.Symbol, RiskFreeInterestRateModel, _dividendYieldProvider,
                            contractB.Symbol, _optionPricingModel
                        );
                }

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var option in chain)
            {
                var indicator = (Securities[option] as dynamic).Rho;
                var underlying = option.Symbol.Underlying;
                var mirror = QuantConnect.Symbol.CreateOption(
                    underlying.Value, option.ID.Market, option.ID.OptionStyle,
                    option.ID.OptionRight == OptionRight.Call ? OptionRight.Put : OptionRight.Call,
                    option.ID.StrikePrice, option.ID.Date
                );

                // Check if price data is available for both contracts and the underlying asset.
                var q = slice.QuoteBars;
                var b = slice.Bars;
                if (q.ContainsKey(option) && q.ContainsKey(mirror) && b.ContainsKey(underlying))
                {
                    var dataPoints = new List<IndicatorDataPoint>
                    {
                        new IndicatorDataPoint(option, q[option].EndTime, q[option].Close),
                        new IndicatorDataPoint(mirror, q[mirror].EndTime, q[mirror].Close),
                        new IndicatorDataPoint(
                            underlying, b[underlying].EndTime, b[underlying].Close
                        )
                    };
                    foreach (var dataPoint in dataPoints)
                    {
                        indicator.Update(dataPoint);
                    }

                    // Get the current value of the Rho indicator.
                    var value = indicator.Current.Value;
                }
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

For more information about the `Rho` constructor, see Using R Indicator .

Volatility Smoothing

The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, pass a `mirrorOption` argument to the indicator method or constructor and then call the

`SetSmoothingFunction` method of the `ImpliedVolatility` property of the indicator.

```C#
// Example: Average IV of the call-put pair.
rhoIndicator.ImpliedVolatility.SetSmoothingFunction((iv, mirrorIv) => (iv + mirrorIv) * 0.5m);
```

Examples

Demonstration Algorithms
OptionIndicatorsRegressionAlgorithm.cs C# OptionIndicatorsMirrorContractsRegressionAlgorithm.cs C#

Equity Options

Market Hours

Introduction

This page shows the trading hours, holidays, and time zone of the Equity Option market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Equity Option market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:00:00
Tuesday	09:30:00 to 16:00:00
Wednesday	09:30:00 to 16:00:00
Thursday	09:30:00 to 16:00:00
Friday	09:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

LEAN uses the trading holidays from the NYSE website.

The following table shows the dates of holidays for the Equity Option market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Equity Option market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/New York)**
1999-11-26	13:00:00
2000-07-03	13:00:00
2000-11-24	13:00:00
2001-07-03	13:00:00
2001-11-23	13:00:00

2001-12-24	13:00:00
2002-07-05	13:00:00
2002-11-29	13:00:00
2002-12-24	13:00:00
2003-07-03	13:00:00
2003-11-28	13:00:00
2003-12-24	13:00:00
2003-12-26	13:00:00
2004-11-26	13:00:00
2005-11-25	13:00:00
2006-07-03	13:00:00
2006-11-24	13:00:00
2007-07-03	13:00:00
2007-11-23	13:00:00
2007-12-24	13:00:00
2008-07-03	13:00:00
2008-11-28	13:00:00
2008-12-24	13:00:00
2009-11-27	13:00:00
2009-12-24	13:00:00
2010-11-26	13:00:00
2011-11-25	13:00:00
2012-07-03	13:00:00
2012-11-23	13:00:00
2012-12-24	13:00:00
2013-07-03	13:00:00
2013-11-29	13:00:00
2013-12-24	13:00:00

2014-07-03	13:00:00
2014-11-28	13:00:00
2014-12-24	13:00:00
2015-11-27	13:00:00
2015-12-24	13:00:00
2016-11-25	13:00:00
2017-07-03	13:00:00
2017-11-24	13:00:00
2017-12-24	13:00:00
2018-07-03	13:00:00
2018-11-23	13:00:00
2018-12-24	13:00:00
2019-07-03	13:00:00
2019-11-29	13:00:00
2019-12-24	13:00:00
2020-11-27	13:00:00
2020-12-24	13:00:00
2021-11-26	13:00:00
2022-11-25	13:00:00
2023-07-03	13:00:00
2023-11-24	13:00:00
2024-07-03	13:00:00
2024-11-29	13:00:00
2024-12-24	13:00:00
2025-07-03	13:00:00
2025-11-28	13:00:00
2025-12-24	13:00:00
2026-11-27	13:00:00

2026-12-24	13:00:00

Late Opens

There are no days with late opens.

Time Zone

The Equity Option market trades in the `America/New York` time zone.

Asset Classes

Crypto

A Cryptocurrency is a digital currency that's secured by cryptography, which makes it nearly impossible to counterfeit or double-spend.

Requesting Data

Handling Data

Holdings

Market Hours

See Also

BasicTemplateCryptoAlgorithm.py
BasicTemplateCryptoAlgorithm.cs

Crypto

Requesting Data

Introduction

Request Crypto data in your algorithm to receive a feed of asset prices in the `OnData` method. For more information about the specific datasets we use for backtests, see the CoinAPI datasets . To trade Crypto live, you can use the QuantConnect data provider .

Create Subscriptions

To create a Crypto subscription, in the `Initialize` method, call the `AddCrypto` method. The `AddCrypto` method returns a `Crypto` object, which contains a `Symbol` property. Save a reference to the `Symbol` so you can use it in `OnData` to access the security data in the `Slice` .

```
C#
_symbol = AddCrypto("BTCUSD").Symbol;
```

The `AddCrypto` method creates a subscription for a single Crypto asset and adds it to your user-defined universe. To create a dynamic universe of Crypto assets, add a Crypto universe .

To view the supported assets, see the CoinAPI datasets .

Fungible Assets

Fungible assets are assets that are interchangeable with each other. For example, every Bitcoin is effectively identical to every other Bitcoin. In contrast, non-fungible tokens (NFTs) are non-fungible because each one is unique. An example of an NFT is the Bored Ape Yacht Club collection. LEAN doesn't currently support trading NFTs.

Resolutions

The following table shows the available resolutions and data formats for Crypto subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`			✓	✓
`Second`	✓	✓		
`Minute`	✓	✓		
`Hour`	✓	✓		
`Daily`	✓	✓		

The default resolution for Crypto subscriptions is `Resolution.Minute` . To change the resolution, pass a `resolution` argument to the `AddCrypto` method.

```
_symbol = AddCrypto("BTCUSD", Resolution.Daily).Symbol;
```

To create custom resolution periods, see Consolidating Data .

Supported Markets

The following `Market` enumeration members are available for Crypto:

To set the market for a security, pass a `market` argument to the `AddCrypto` method.

```
_symbol = AddCrypto("BTCUSD", market: Market.Coinbase).Symbol;
```

The brokerage models have a default market for each asset class. If you set a brokerage model, you may not need to specify the market to use.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
_symbol = AddCrypto("BTCUSD", fillForward: false).Symbol;
```

Margin and Leverage

LEAN models buying power and margin calls to ensure your algorithm stays within the margin requirements. The amount of leverage available to you depends on the brokerage you use. To change the amount of leverage you can use for a security, pass a `leverage` argument to the `AddCrypto` method.

```
_symbol = AddCrypto("BTCUSD", leverage: 3).Symbol;
```

For more information about the leverage each brokerage provides, see Brokerages .

Data Normalization

The data normalization mode doesn't affect the data that LEAN passes to `OnData` or the data from history request . If you change the data normalization mode, it won't change the outcome.

Properties

The `AddCrypto` method returns a `Crypto` object, which have the following properties:

Examples

The following examples demonstrate some common practices for requesting Crypto data.

Example 1: Initialize Binance Algorithms

The following algorithm requests data for the BTC/USDT and ETH/USDT Crypto pairs on Binance. The Binance brokerage model provides realistic fee and fill modeling for the backtest. Since the quote currency of both pairs is USDT, set the account

currency to USDT instead of USD. Otherwise, some order will fail since the brokerage doesn't automatically convert USD to USDT for you.

```C#
public class CryptoExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Set the brokerage and account type for accurate fee and margin behavior.
        SetBrokerageModel(BrokerageName.Binance, AccountType.Cash);

        // In some Crypto brokerages, USD is not a valid currency to trade.
        // Therefore, set the account currency to USDT and add the starting cash.
        SetAccountCurrency("USDT", 100000);

        // Subscribe to the Crypto pairs.
        // You don't need to specify the market because the Binance brokerage model already does that.
        AddCrypto("BTCUSDT");
        AddCrypto("ETHUSDT");
    }
}
```

Crypto

Handling Data

Introduction

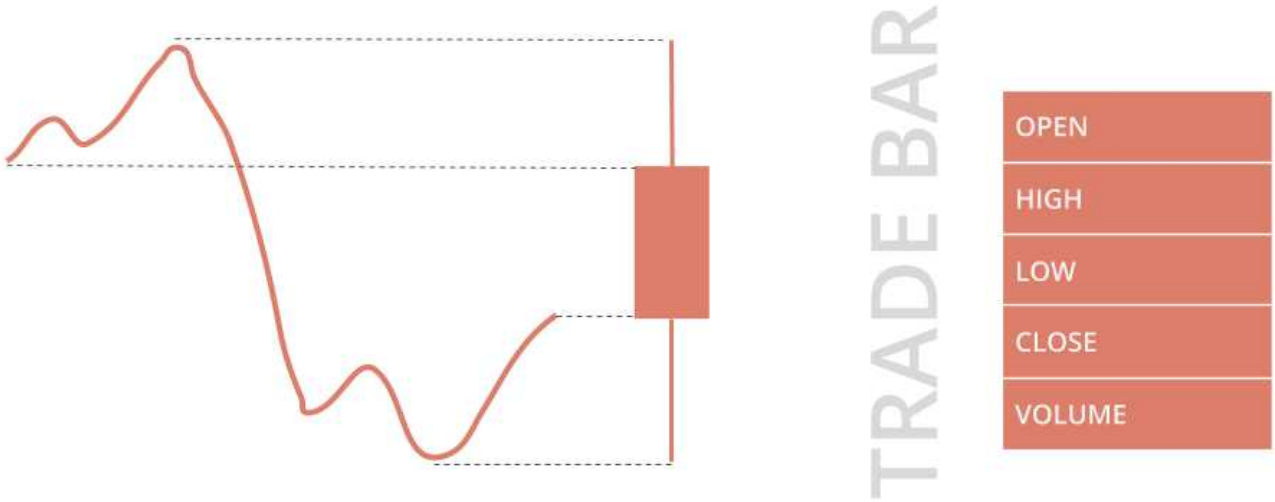
LEAN passes the data you request to the `OnData` method so you can make trading decisions. The default `OnData` method accepts a `Slice` object, but you can define additional `OnData` methods that accept different data types. For example, if you define an `OnData` method that accepts a `TradeBar` argument, it only receives `TradeBar` objects. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `TradeBars` `DataDictionary` is made up of `TradeBar` objects. To access individual data points in the dictionary, you can index the dictionary with the security ticker or `Symbol` , but we recommend you use the `Symbol` .

To view the resolutions that are available for Crypto data, see Resolutions .

Trades

`TradeBar` objects are price bars that consolidate individual trades from the exchanges. They contain the open, high, low, close, and volume of trading activity over a period of time.



To get the `TradeBar` objects in the `Slice` , index the `Slice` or index the `Bars` property of the `Slice` with the security `Symbol` . If the security doesn't actively trade or you are in the same time step as when you added the security subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your security before you index the `Slice` with the security `Symbol` .

```C#
public override void OnData(Slice slice)
{
    // Check if the symbol is contained in TradeBars object
    if (slice.Bars.ContainsKey(_symbol))
    {
        // Obtain the mapped TradeBar of the symbol
        var tradeBar = slice.Bars[_symbol];
    }
}
```

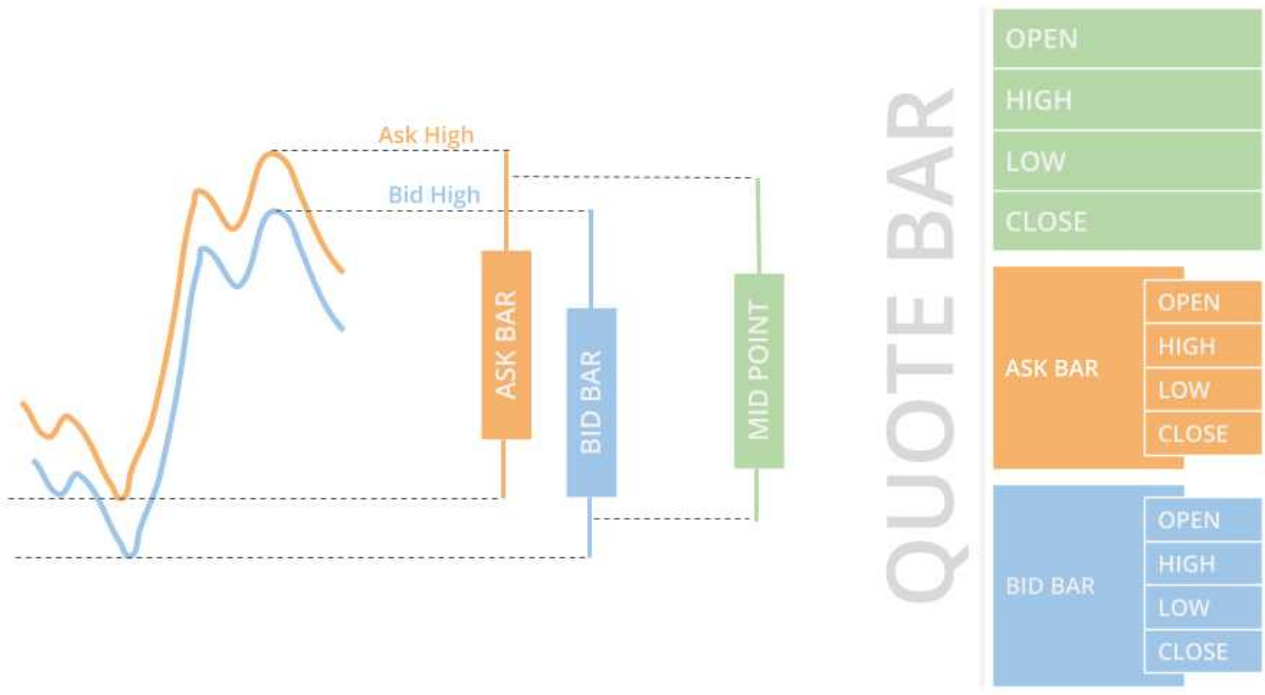
You can also iterate through the `TradeBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `TradeBar` objects.

```C#
public override void OnData(Slice slice)
{
    // Iterate all received Symbol-TradeBar key-value pairs
    foreach (var kvp in slice.Bars)
    {
        var symbol = kvp.Key;
        var tradeBar = kvp.Value;
        var closePrice = tradeBar.Close;
    }
}
```

`TradeBar` objects have the following properties:

Quotes

`QuoteBar` objects are bars that consolidate NBBO quotes from the exchanges. They contain the open, high, low, and close prices of the bid and ask. The `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` object are the mean of the respective bid and ask prices. If the bid or ask portion of the `QuoteBar` has no data, the `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` copy the values of either the `Bid` or `Ask` instead of taking their mean.



To get the `QuoteBar` objects in the `Slice` , index the `QuoteBars` property of the `Slice` with the security `Symbol` . If the security doesn't actively get quotes or you are in the same time step as when you added the security subscription, the `Slice` may not

contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your security before you index the `Slice` with the security `Symbol` .

```
C#
public override void OnData(Slice slice)
{
    // Check if the symbol is contained in QuoteBars object
    if (slice.QuoteBars.ContainsKey(_symbol))
    {
        // Obtain the mapped QuoteBar of the symbol
        var quoteBar = slice.QuoteBars[_symbol];
    }
}
```

You can also iterate through the `QuoteBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `QuoteBar` objects.

```
C#
public override void OnData(Slice slice)
{
    // Iterate all received Symbol-QuoteBar key-value pairs
    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        var askPrice = quoteBar.Ask.Close;
    }
}
```

`QuoteBar` objects let LEAN incorporate spread costs into your simulated trade fills to make backtest results more realistic.

`QuoteBar` objects have the following properties:

Ticks

`Tick` objects represent a single trade or quote at a moment in time. A trade tick is a record of a transaction for the security. A quote tick is an offer to buy or sell the security at a specific price.

Trade ticks have a non-zero value for the `Quantity` and `Price` properties, but they have a zero value for the `BidPrice` , `BidSize` , `AskPrice` , and `AskSize` properties. Quote ticks have non-zero values for `BidPrice` and `BidSize` properties or have non-zero values for `AskPrice` and `AskSize` properties. To check if a tick is a trade or a quote, use the `TickType` property.

In backtests, LEAN groups ticks into one millisecond buckets. In live trading, LEAN groups ticks into ~70-millisecond buckets. To get the `Tick` objects in the `Slice` , index the `Ticks` property of the `Slice` with a `Symbol` . If the security doesn't actively trade or you are in the same time step as when you added the security subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your security before you index the `Slice` with the security `Symbol` .

```
C#
public override void OnData(Slice slice)
{
    if (slice.Ticks.ContainsKey(_symbol))
    {
        var ticks = slice.Ticks[_symbol];
        foreach (var tick in ticks)
        {
            var price = tick.Price;
        }
    }
}
```

You can also iterate through the `Ticks` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the

`List<Tick>` objects.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Ticks)
    {
        var symbol = kvp.Key;
        var ticks = kvp.Value;
        foreach (var tick in ticks)
        {
            var price = tick.Price;
        }
    }
}
```

Tick data is raw and unfiltered, so it can contain bad ticks that skew your trade results. For example, some ticks come from dark pools, which aren't tradable. We recommend you only use tick data if you understand the risks and are able to perform your own online tick filtering.

`Tick` objects have the following properties:

Examples

The following examples demonstrate some common practices for handling Crypto data.

Example 1: Dollar Cost Average BTC

Dollar cost averaging (DCA) is where you consistently invest a fixed dollar amount into an asset on a regular basis (for example, monthly), regardless of the asset's price. It can reduce the volatility in your PnL due to slowly increasing the position size over time. The following algorithm demonstrates a DCA investment into BTC. It buys $3,000 USD worth of BTC every midnight for 30 consecutive days.

```C#
public class CryptoExampleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    // Set a day count variable for counting the days of the DCA trade.
    private int _dayCount = 0;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Set the brokerage and account type to match your brokerage environment for accurate fee and margin
behavior.
        SetBrokerageModel(BrokerageName.Bitfinex, AccountType.Cash);
        // For daily DCA purchases, subscribe to daily asset data.
        _symbol = AddCrypto("BTCUSD", Resolution.Daily).Symbol;
    }

    public override void OnData(Slice slice)
    {
        // If you haven't invested for 30 consecutive days yet, continue buying.
        if (slice.Bars.ContainsKey(_symbol) && _dayCount++ < 30)
        {
            // Calculate the order size for $3,000 USD using the current price.
            var quantity = 3000m / slice.Bars[_symbol].Close;
            // Round down to observe the lot size.
            var lotSize = Securities[_symbol].SymbolProperties.LotSize;
            quantity = (long)(quantity / lotSize) * lotSize;
            if (quantity != 0)
            {
                MarketOrder(_symbol, quantity);
            }
        }
    }
}
```

Example 2: Inter-Market Spread

There is always a small discrepancy in the price of the same Crypto pair trading on different exchanges. Although you can't currently live trade on algorithm with multiple brokerages, you can study the cointegration pattern and implement 2 live nodes to capture the arbitrage opportunity. The following algorithm demonstrates how to obtain the spread between the BTCUSD pair on Kraken and on Coinbase:

```C#
public class CryptoExampleAlgorithm : QCAlgorithm
{
    private Symbol _krakenBtc;
    private Symbol _coinbaseBtc;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Subscribe to BTC/USD on 2 different exchanges.
        _krakenBtc = AddCrypto("BTCUSD", market: Market.Kraken).Symbol;
        _coinbaseBtc = AddCrypto("BTCUSD", market: Market.Coinbase).Symbol;
    }

    public override void OnData(Slice slice)
    {
        // Only calculate the spread if the prices on both exchanges are in the current Slice.
        if (slice.Bars.ContainsKey(_krakenBtc) && slice.Bars.ContainsKey(_coinbaseBtc))
        {
            // Calculate the spread between the 2 exchanges, making sure the comparison is always in the same
direction.
            var spread = slice.Bars[_krakenBtc].Close - slice.Bars[_coinbaseBtc].Close;
        }
    }
}
```

Crypto

Holdings

Introduction

Crypto holdings require special consideration because not all brokerages track your positions in each currency pair.

Cash vs Margin Brokerage Accounts

Some of the Crypto brokerages integrated into QuantConnect support cash and margin accounts while some only support cash accounts. If you want to short Crypto assets or use leverage, you need a margin account. Follow these steps to view which account types each brokerage supports:

1. Open the brokerage guides .
2. Click a Crypto brokerage.
3. Scroll down to the Account Types section.

Virtual Pairs

All fiat and Crypto currencies are individual assets. When you buy a pair like BTCUSD, you trade USD for BTC. In this case, LEAN removes some USD from your portfolio cashbook and adds some BTC. The virtual pair BTCUSD represents your position in that trade, but the virtual pair doesn't actually exist. It simply represents an open trade. If you call the Liquidate method with the `Symbol` of a virtual pair, the method only liquidates the quantity in your virtual pair. For more information about liquidating Crypto postitions, see Crypto Trades .

Access Cypto Holdings

The `CashBook` stores the quantity of your Crypto holdings. To access your Crypto holdings, index the `CashBook` with the currency ticker. The values of the `CashBook` dictionary are `Cash` objects, which have the following properties:

```
C#
var btcQuantity = Portfolio.CashBook["BTC"].Amount;
var btcValue = Portfolio.CashBook["BTC"].ValueInAccountCurrency;
```

To access the virtual pair of your Crypto trades, index the `Portfolio` object with the pair `Symbol` .

```
C#
var securityHolding = Portfolio[_btcUsdSymbol];
```

Stateful Redeployments

When you make a trade inside of a running algorithm, LEAN tracks the virtual position state for you, but it won't survive between deployments. Some brokerages save your virtual pairs, so you can load them into your algorithm when you stop and redeploy it. Depending on the brokerage you select, you may be able to manually add virtual pairs when you deploy live algorithms with the LEAN CLI or cloud deployment wizard .

Examples

The following examples demonstrate some common practices for crypto holdings.

Example 1: ETH-BTC Proxy Trading

The following algorithm trades trend-following of the ETHBTC crypto pair using a 20-day EMA indicator. To reduce friction (e.g. slippage), we trade the more liquid and popular BTCUSDT and ETHUSDT pair instead. For example, if ETHBTC is above EMA (uptrend), we sell all BTCUSDT holdings and buy ETHUSDT with the same USDT value, going flat on BTC while holding ETH.

C#

```
public class CryptoHoldingsAlgorithm : QCAlgorithm
{
    private Symbol _btcusdt, _ethusdt, _ethbtc;
    private ExponentialMovingAverage _ema;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetAccountCurrency("USDT", 1000000m);

        // We would like to trade the EMA cross between 2 popular cryptos: BTC & ETH,
        // so request ETHBTC data to find trading opportunities.
        _ethbtc = AddCrypto("ETHBTC", Resolution.Daily, market: Market.Coinbase).Symbol;
        // Trade through BTCUSDT & ETHUSDT, though, since stable coin trades have lower friction costs and higher
liquidity.
        var btcusdt = AddCrypto("BTCUSDT", Resolution.Daily, market: Market.Coinbase);
        var ethusdt = AddCrypto("ETHUSDT", Resolution.Daily, market: Market.Coinbase);
        _btcusdt = btcusdt.Symbol;
        _ethusdt = ethusdt.Symbol;

        // Simulate an account with various crypto cash through holdings.
        btcusdt.Holdings.SetHoldings(btcusdt.Price, 5m);
        btcusdt.BaseCurrency.AddAmount(5m);
        ethusdt.Holdings.SetHoldings(ethusdt.Price, 22.5m);
        ethusdt.BaseCurrency.AddAmount(22.5m);

        // Add automatic updating of the EMA indicator for trend trade signal emission.
        _ema = EMA(_ethbtc, 20, Resolution.Daily);
        // Warm up the indicator for its readiness usage immediately.
        WarmUpIndicator(_ethbtc, _ema, Resolution.Daily);
    }

    public override void OnData(Slice slice)
    {
        if (slice.Bars.TryGetValue(_ethbtc, out var bar) && _ema.IsReady &&
            slice.Bars.TryGetValue(_btcusdt, out var btc) && slice.Bars.TryGetValue(_ethusdt, out var eth))
        {
            var ema = _ema.Current.Value;
            var btcHoldings = Portfolio.CashBook["BTC"].Amount;
            var ethHoldings = Portfolio.CashBook["ETH"].Amount;

            // ETHBTC's current price is higher than EMA, suggesting an uptrend.
            if (bar.Close > ema && !Portfolio[_btcusdt].IsShort)
            {
                // To follow the up trend of ETHBTC, sell ALL BTCUSDT and buy ETHUSDT with the same value.
                if (btcHoldings > 0)
                {
                    var btcValue = btcHoldings * btc.Close;
                    MarketOrder(_btcusdt, -btcHoldings);
                    var ethSize = btcValue / eth.Close;
                    MarketOrder(_ethusdt, ethSize);
                }
            }
            // ETHBTC's current price is below the EMA, suggesting a downtrend.
            else if (bar.Close < ema && !Portfolio[_btcusdt].IsLong)
            {
                // To follow the downtrend of ETHBTC, sell ALL ETHUSDT and buy BTCUSDT with the same value.
                if (ethHoldings > 0)
                {
                    var ethValue = ethHoldings * eth.Close;
                    MarketOrder(_ethusdt, -ethHoldings);
                    var btcSize = ethValue / btc.Close;
                    MarketOrder(_btcusdt, btcSize);
                }
            }
        }
    }
}
```

Crypto

Market Hours

Introduction

The Crypto markets are open 24/7. There are no pre-market hours, post-market hours, holidays, late opens, or early closes.

Asset Classes

Crypto Futures

Crypto Perpetual Futures are derivative financial contracts that obligate parties to buy or sell an asset at an unspecified future date.

Requesting Data

Handling Data

Market Hours

See Also

BasicTemplateCryptoFutureAlgorithm.py
BasicTemplateCryptoFutureAlgorithm.cs

Crypto Futures

Requesting Data

Introduction

Request Crypto Futures data in your algorithm to receive a feed of asset prices in the `OnData` method. For more information about the specific datasets we use for backtests, see the Binance Crypto Future Price Data dataset listing . To trade Crypto Futures live, you can use the QuantConnect data provider .

Create Subscriptions

To create a Crypto Futures subscription, in the `Initialize` method, call the `AddCryptoFuture` method. The `AddCryptoFuture` method returns a `CryptoFuture` object, which contains a `Symbol` property. Save a reference to the `Symbol` so you can use it in `OnData` to access the security data in the `Slice` .

```
C#
_symbol = AddCryptoFuture("BTCUSD").Symbol;
```

The `AddCryptoFuture` method creates a subscription for a single Crypto Futures asset and adds it to your user-defined universe.

To view the supported assets in the Crypto Futures dataset, see Supported Assets .

Resolutions

The following table shows the available resolutions and data formats for Crypto Futures contract subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`			✓	✓
`Second`	✓	✓		
`Minute`	✓	✓		
`Hour`	✓	✓		
`Daily`	✓	✓		

The default resolution for Crypto Futures subscriptions is `Resolution.Minute` . To change the resolution, pass a `resolution` argument to the `AddCryptoFuture` method.

```
C#
_symbol = AddCryptoFuture("BTCUSD", Resolution.Daily).Symbol;
```

To create custom resolution periods, see Consolidating Data .

Supported Markets

The following `Market` enumeration members are available for Cryptofuture:

```
_symbol = AddCryptoFuture("BTCUSD", market: Market.Binance).Symbol;
```

The brokerage models have a default market for each asset class. If you set a brokerage model, you may not need to specify the market to use.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
_symbol = AddCryptoFuture("BTCUSD", fillForward: false).Symbol;
```

Margin and Leverage

LEAN models buying power and margin calls to ensure your algorithm stays within the margin requirements. The amount of leverage available to you depends on the brokerage you use. To change the amount of leverage you can use for a security, pass a `leverage` argument to the `AddCryptoFuture` method.

```
_symbol = AddCryptoFuture("BTCUSD", leverage: 3).Symbol;
```

For more information about the leverage each brokerage provides, see Brokerages .

Data Normalization

The data normalization mode doesn't affect the data that LEAN passes to `OnData` or the data from history request . If you change the data normalization mode, it won't change the outcome.

Properties

The `AddCryptoFuture` method returns a `CryptoFuture` object, which have the following properties:

Examples

The following examples demonstrate some common practices for requesting Crypto Futures data.

Example 1: Respect Lot Sizes

Like Futures, Crypto Futures contracts have a fixed discrete lot size to trade instead of fractional like spot Crypto pairs. The following algorithm demonstrates how to place orders that respect the lot size to avoid rounding errors.

C#

```
public class CryptoFutureAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Set brokerage and account type to match your brokerage environment for accurate fee and margin behavior.
        SetBrokerageModel(BrokerageName.Binance, AccountType.Margin);
        // In the Binance brokerage, you can't trade with USD.
        // Set the account currency as USDT and add the starting cash.
        SetAccountCurrency("USDT", 100000);
        // Subscribe to the BTCUSDT perpetual Future contract.
        _symbol = AddCryptoFuture("BTCUSDT").Symbol;
    }

    public override void OnData(Slice slice)
    {
        // Only place orders when market is open since market on open orders aren't supported.
        if (!Portfolio.Invested && IsMarketOpen(_symbol))
        {
            // Get the lot size from the symbol properties. Placing an order that respects the lot size
            // ensures the order is valid and allows accurate profit and risk calculations for the whole portfolio.
            var lotSize = Securities[_symbol].SymbolProperties.LotSize;
            // This example demonstrates an initial desired order size of 2.5 contracts, which will be rounded
            // to 2 contracts since the lot size is 1.
            var quantity = Convert.ToDecimal(Math.Floor(2.5 / (double)lotSize)) * lotSize;
            MarketOrder(_symbol, quantity);
        }
    }
}
```

Crypto Futures

Handling Data

Introduction

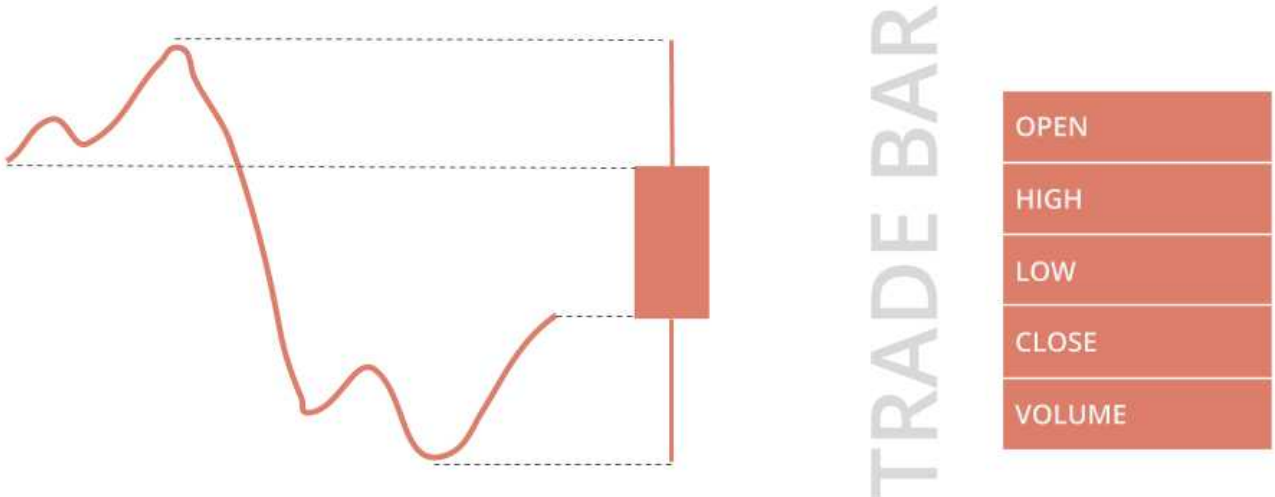
LEAN passes the data you request to the `OnData` method so you can make trading decisions. The default `OnData` method accepts a `Slice` object, but you can define additional `OnData` methods that accept different data types. For example, if you define an `OnData` method that accepts a `TradeBar` argument, it only receives `TradeBar` objects. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `TradeBars` `DataDictionary` is made up of `TradeBar` objects. To access individual data points in the dictionary, you can index the dictionary with the security ticker or `Symbol` , but we recommend you use the `Symbol` .

To view the resolutions that are available for Crypto Futures data, see Resolutions .

Trades

`TradeBar` objects are price bars that consolidate individual trades from the exchanges. They contain the open, high, low, close, and volume of trading activity over a period of time.



To get the `TradeBar` objects in the `Slice` , index the `Slice` or index the `Bars` property of the `Slice` with the security `Symbol` . If the security doesn't actively trade or you are in the same time step as when you added the security subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your security before you index the `Slice` with the security `Symbol` .

```C#
public override void OnData(Slice slice)
{
    // Check if the symbol is contained in TradeBars object
    if (slice.Bars.ContainsKey(_symbol))
    {
        // Obtain the mapped TradeBar of the symbol
        var tradeBar = slice.Bars[_symbol];
    }
}
```

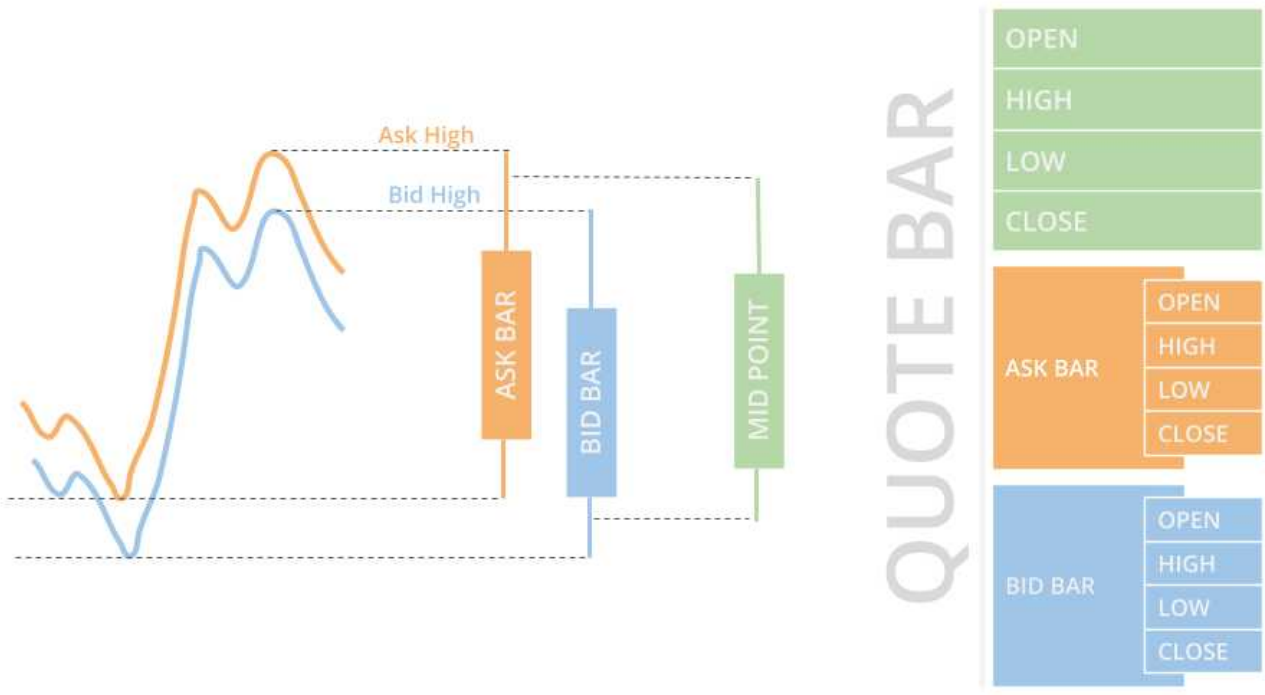
You can also iterate through the `TradeBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `TradeBar` objects.

```C#
public override void OnData(Slice slice)
{
    // Iterate all received Symbol-TradeBar key-value pairs
    foreach (var kvp in slice.Bars)
    {
        var symbol = kvp.Key;
        var tradeBar = kvp.Value;
        var closePrice = tradeBar.Close;
    }
}
```

`TradeBar` objects have the following properties:

Quotes

`QuoteBar` objects are bars that consolidate NBBO quotes from the exchanges. They contain the open, high, low, and close prices of the bid and ask. The `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` object are the mean of the respective bid and ask prices. If the bid or ask portion of the `QuoteBar` has no data, the `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` copy the values of either the `Bid` or `Ask` instead of taking their mean.



To get the `QuoteBar` objects in the `Slice` , index the `QuoteBars` property of the `Slice` with the security `Symbol` . If the security doesn't actively get quotes or you are in the same time step as when you added the security subscription, the `Slice` may not

contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your security before you index the `Slice` with the security `Symbol` .

```
C#
public override void OnData(Slice slice)
{
    // Check if the symbol is contained in QuoteBars object
    if (slice.QuoteBars.ContainsKey(_symbol))
    {
        // Obtain the mapped QuoteBar of the symbol
        var quoteBar = slice.QuoteBars[_symbol];
    }
}
```

You can also iterate through the `QuoteBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `QuoteBar` objects.

```
C#
public override void OnData(Slice slice)
{
    // Iterate all received Symbol-QuoteBar key-value pairs
    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        var askPrice = quoteBar.Ask.Close;
    }
}
```

`QuoteBar` objects let LEAN incorporate spread costs into your simulated trade fills to make backtest results more realistic.

`QuoteBar` objects have the following properties:

Ticks

`Tick` objects represent a single trade or quote at a moment in time. A trade tick is a record of a transaction for the security. A quote tick is an offer to buy or sell the security at a specific price.

Trade ticks have a non-zero value for the `Quantity` and `Price` properties, but they have a zero value for the `BidPrice` , `BidSize` , `AskPrice` , and `AskSize` properties. Quote ticks have non-zero values for `BidPrice` and `BidSize` properties or have non-zero values for `AskPrice` and `AskSize` properties. To check if a tick is a trade or a quote, use the `TickType` property.

In backtests, LEAN groups ticks into one millisecond buckets. In live trading, LEAN groups ticks into ~70-millisecond buckets. To get the `Tick` objects in the `Slice` , index the `Ticks` property of the `Slice` with a `Symbol` . If the security doesn't actively trade or you are in the same time step as when you added the security subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your security before you index the `Slice` with the security `Symbol` .

```
C#
public override void OnData(Slice slice)
{
    if (slice.Ticks.ContainsKey(_symbol))
    {
        var ticks = slice.Ticks[_symbol];
        foreach (var tick in ticks)
        {
            var price = tick.Price;
        }
    }
}
```

You can also iterate through the `Ticks` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the

`List<Tick>` objects.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Ticks)
    {
        var symbol = kvp.Key;
        var ticks = kvp.Value;
        foreach (var tick in ticks)
        {
            var price = tick.Price;
        }
    }
}
```

Tick data is raw and unfiltered, so it can contain bad ticks that skew your trade results. For example, some ticks come from dark pools, which aren't tradable. We recommend you only use tick data if you understand the risks and are able to perform your own online tick filtering.

`Tick` objects have the following properties:

Margin Interest Rates

`MarginInterestRate` objects contain the margin interest rate, which is a cost associated with trading on margin. To get the `MarginInterestRate` objects in the `Slice` , index the `MarginInterestRates` property of the `Slice` with the Crypto Future `Symbol` . The `MarginInterestRates` property of the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the property contains data for your Crypto Future before you index it with the Crypto Future `Symbol` .

```
C#
public override void OnData(Slice slice)
{
    if (slice.MarginInterestRates.TryGetValue(_symbol, out var marginInterestRate))
    {
        var interestRate = marginInterestRate.InterestRate;
    }
}
```

You can also iterate through the `MarginInterestRates` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `MarginInterestRate` objects.

```
C#
public override void OnData(Slice slice)
{
    foreach (var (symbol, marginInterestRate) in slice.MarginInterestRates)
    {
        var interestRate = marginInterestRate.InterestRate;
    }
}
```

`MarginInterestRate` objects have the following properties:

Examples

The following examples demonstrate some common practices for handling Crypto Futures data.

Example 1: Adjust Holdings on Margin Payments

Perpetual futures have a mechanism called funding , where if you're holding a position at certain times (the funding timestamp), you might have to pay or receive funding based on the difference between the perpetual contract price and the spot price. This

example demonstrates how to adjust your position in Crypto Future based on the funding rate. The funding is settled in the quote currency, which is USDT in this case. When you receive USDT, the algorithm increases its position size in BTCUSDT. When you pay USDT, the algorithm decreases its position size.

```C#
public class CryptoFutureAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private decimal _lotSize;
    private int _day = -1;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Set brokerage and account type to match your brokerage environment for accurate fee and margin behavior.
        SetBrokerageModel(BrokerageName.Binance, AccountType.Margin);
        // In the Binance brokerage, you can't trade with USD.
        // Set the account currency as USDT and add the starting cash.
        SetAccountCurrency("USDT", 1000000);
        // Subscribe to the BTCUSDT perpetual Future contract.
        var btcusdt = AddCryptoFuture("BTCUSDT");
        _symbol = btcusdt.Symbol;
        // Save the lot size to avoid order errors.
        _lotSize = btcusdt.SymbolProperties.LotSize;
        // Set the starting BTC balance to 10.
        btcusdt.BaseCurrency.SetAmount(10);
    }

    public override void OnData(Slice slice)
    {
        // Only place orders when the market is open since market on open orders aren't supported.
        if (!Portfolio.Invested && IsMarketOpen(_symbol))
        {
            // Open a long position in the perpetual Crypto Future.
            MarketOrder(_symbol, 10);
        }
        // Only run the following logic once per day.
        if (_day == Time.Day)
        {
            return;
        }
        // Get the current margin interest rate.
        MarginInterestRate interestRate;
        if (!slice.MarginInterestRates.TryGetValue(_symbol, out interestRate))
        {
            return;
        }
        // Calculate the funding payment.
        var holding = Portfolio[_symbol];
        var positionValue = holding.GetQuantityValue(holding.Quantity).Amount;
        var funding = interestRate.InterestRate * positionValue / holding.Price;
        // Increase/decrease the position size based on the funding payment.
        var quantity = -(int)(funding / _lotSize) * _lotSize;
        if (quantity != 0)
        {
            MarketOrder(_symbol, quantity);
            // Plot the portfolio state.
            Plot("CashBook", "USDT", Portfolio.CashBook["USDT"].Amount);
            Plot("CashBook", "BTC", Portfolio.CashBook["BTC"].Amount);
            Plot("Quantity", "BTCUSDT", Portfolio[_symbol].Quantity);
        }
        _day = Time.Day;
    }
}
```

Example 2: Future-Spot Arbitrage

Long-short arbitrage involves simultaneously trading BTCUSDT in the spot market and BTCUSDT Futures with the same size but in the opposite direction. The following algorithm monitors the spread between these markets. If the spot price exceeds the Future price by a threshold, it shorts the spot market and buys the Future contract (and vice vera for the other way around), assuming their discrepancies will be wiped out within a short period due to market efficiency. It closes the positions after the

spread inverts.

C#

```
public class CryptoFutureArbitrageAlgorithm : QCAlgorithm
{
    private Crypto _spot;
    private CryptoFuture _future;
    private decimal _multiplier;
    private decimal _threshold = 1.005m;  // 0.5%

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Set brokerage and account type to match your brokerage
        // environment for accurate fee and margin behavior.
        SetBrokerageModel(BrokerageName.Binance, AccountType.Margin);
        // In the Binance brokerage, you can't trade with USD.
        // Set the account currency to USDT and add the starting cash.
        SetAccountCurrency("USDT", 50000000);
        // Add the BTCUSDT spot and Futures markets.
        _spot = AddCrypto("BTCUSDT", fillForward: false, market: Market.Binance);
        _future = AddCryptoFuture("BTCUSDT", fillForward: false, market: Market.Binance);
        _multiplier = _future.SymbolProperties.ContractMultiplier;
    }

    public override void OnData(Slice slice)
    {
        // Wait for there to be a TradeBar for each asset, so we know
        // the prices are up-to-date.
        foreach (var security in new List<Security>() {_spot, _future})
            if (!slice.Bars.ContainsKey(security.Symbol)) return;
        var spot = _spot.Price;
        var future = _future.Price;
        // Buy low sell high: If one's price is above another by the
        // set threshold, sell it and buy the other security. The
        // threshold ensures there is enough profit potential to overcome
        // the fees and slippage. Make sure to equalize the actual order
        // size with the contract multiplier.
        if (!_spot.Invested)
        {
            if (spot >= future * _threshold)
            {
                Enter(-1, 1);
            }
            else if (future >= spot * _threshold)
            {
                Enter(1, -1);
            }
            return;
        }
        // When the mispricing converges, close both positions to earn
        // the spread.
        if ((_spot.Holdings.IsShort && spot < future) ||
            (_spot.Holdings.IsLong && spot > future))
        {
            // Liquidate the spot position.
            var quantity = Math.Min(_spot.Holdings.Quantity, _spot.BaseCurrency.Amount);
            if (quantity < 0m)
            {
                quantity /= (1 - 0.001m);  // To cover the taker fees.
            }
            var lotSize = _spot.SymbolProperties.LotSize;
            quantity = -Math.Round(quantity / lotSize) * lotSize;
            MarketOrder(_spot.Symbol, quantity, tag: "Liquidate Spot");
            // Liquidate the Future position.
            Liquidate(_future.Symbol, tag: "Liquidate Future");
        }
    }

    private void Enter(int spotSign, int futureSign)
    {
        var quantity = CalculateOrderQuantity(_future.Symbol, 0.5d);
        if (quantity == 0) return;
        // Enter the spot position.
        MarketOrder(_spot.Symbol, spotSign * quantity * _multiplier, tag: "Spot");
        // Enter the Future position.
        MarketOrder(_future.Symbol, futureSign * quantity, tag: "Future");
    }
}
```

Crypto Futures

Market Hours

Introduction

The Crypto Futures markets are open 24/7. There are no pre-market hours, post-market hours, holidays, late opens, or early closes.

Asset Classes

Forex

Forex markets are for trading foreign currencies.

Requesting Data

Handling Data

Market Hours

See Also

BasicTemplateForexAlgorithm.py
BasicTemplateForexAlgorithm.cs

Forex

Requesting Data

Introduction

Request Forex data in your algorithm to receive a feed of exchange rates in the `OnData` method. For more information about the specific dataset we use for backtests, see the FOREX dataset listing . To trade Forex live, you can use the QuantConnect data provider .

Create Subscriptions

To create a Forex pair subscription, in the `Initialize` method, call the `AddForex` method. The `AddForex` method returns a `Forex` object, which contains a `Symbol` property. Save a reference to the `Symbol` so you can use it in `OnData` to access the security data in the `Slice` .

```
C#
_symbol = AddForex("EURUSD").Symbol;
```

To view the supported Forex pairs, see Supported Assets .

Resolutions

The following table shows the available resolutions and data formats for Forex subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`				✓
`Second`		✓		
`Minute`		✓		
`Hour`		✓		
`Daily`		✓		

The default resolution for Forex subscriptions is `Resolution.Minute` . To change the resolution, pass a `resolution` argument to the `AddForex` method.

```
C#
_symbol = AddForex("EURUSD", Resolution.Daily).Symbol;
```

To create custom resolution periods, see Consolidating Data .

Supported Markets

The only market available for Forex pairs is `Market.Oanda` , so you don't need to pass a `market` argument to the `AddForex` method.

```
_symbol = AddForex("EURUSD", market: Market.Oanda).Symbol;
```

The brokerage models have a default market for each asset class. If you set a brokerage model, you may not need to specify the market to use.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
_symbol = AddForex("EURUSD", fillForward: false).Symbol;
```

Margin and Leverage

LEAN models buying power and margin calls to ensure your algorithm stays within the margin requirements. The OANDA brokerage let's you use up to 50x leverage on margin accounts. To change the amount of leverage you can use for a Forex pair, pass a `leverage` argument to the `AddForex` method.

```
_symbol = AddForex("EURUSD", leverage: 35).Symbol;
```

Data Normalization

The data normalization mode doesn't affect the data that LEAN passes to `OnData` or the data from history request . If you change the data normalization mode, it won't change the outcome.

Properties

The `AddForex` method returns a `Forex` object, which have the following properties:

Examples

The following examples demonstrate some common practices for requesting Forex data.

Example 1: Add All Forex Pairs

The following algorithm adds all the Forex pairs that trade on the OANDA exchange . To get all of the pairs from the Symbol Properties Database , call the `SymbolPropertiesDatabase.GetSymbolPropertiesList` method. Adding all of the pairs is particularly useful for spotting arbitrage opportunities among the pairs.

C#

```
public class ForexExampleAlgorithm : QCAlgorithm
{
    private List<Forex> _forexPairs;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Get all of the OANDA Forex pairs from the Symbol Properties Database.
        _forexPairs = SymbolPropertiesDatabase.GetSymbolPropertiesList(Market.Oanda, SecurityType.Forex)
            // Add all of the pairs. You don't need to specify the market because the Symbol object already does.
            .Select(x => AddForex(x.Key.Symbol, Resolution.Daily))
            .ToList();
    }
}
```

Forex

Handling Data

Introduction

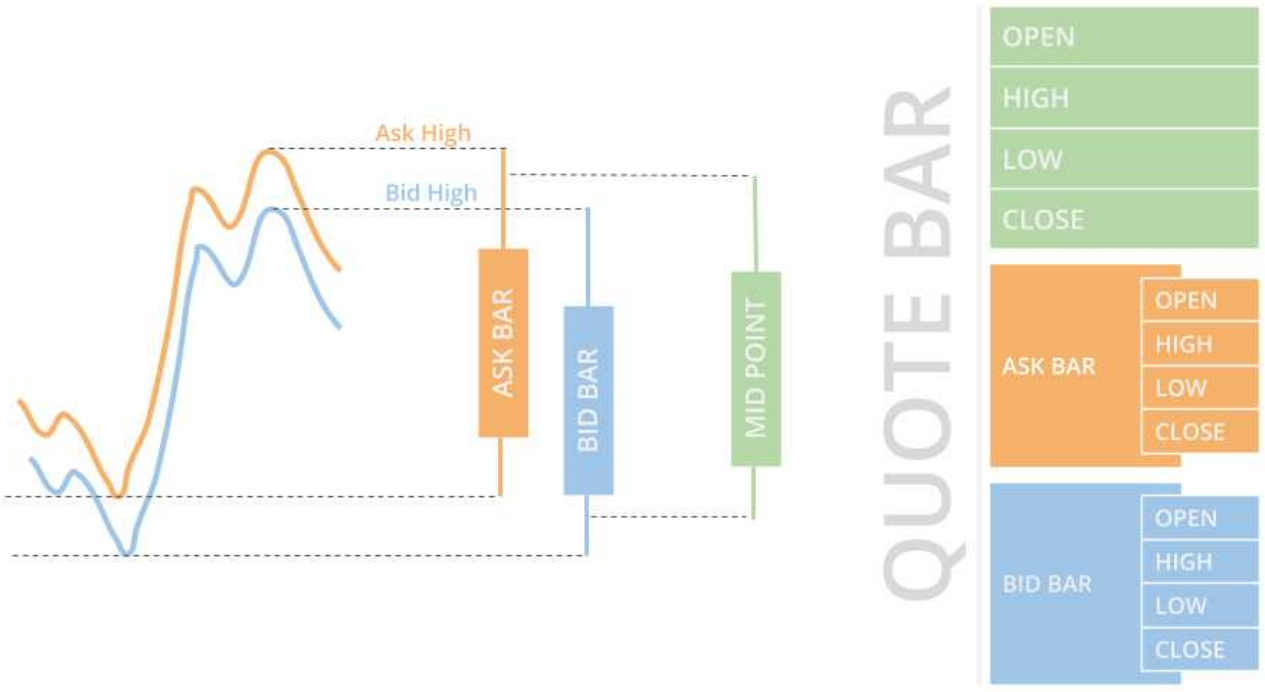
LEAN passes the data you request to the `OnData` method so you can make trading decisions. The default `OnData` method accepts a `Slice` object, but you can define additional `OnData` methods that accept different data types. For example, if you define an `OnData` method that accepts a `QuoteBar` argument, it only receives `QuoteBar` objects. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `QuoteBars DataDictionary` is made up of `QuoteBar` objects. To access individual data points in the dictionary, you can index the dictionary with the Forex pair ticker or `Symbol` , but we recommend you use the `Symbol` .

To view the resolutions that are available for Forex data, see Resolutions .

Quotes

`QuoteBar` objects are bars that consolidate NBBO quotes from the exchanges. They contain the open, high, low, and close prices of the bid and ask. The `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` object are the mean of the respective bid and ask prices. If the bid or ask portion of the `QuoteBar` has no data, the `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` copy the values of either the `Bid` or `Ask` instead of taking their mean.



To get the `QuoteBar` objects in the `Slice` , index the `QuoteBars` property of the `Slice` with the Forex pair `Symbol` . If the Forex pair doesn't actively get quotes or you are in the same time step as when you added the Forex pair subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your Forex pair before you index the

`Slice` with the Forex pair `Symbol` .

```
C#
public override void OnData(Slice slice)
{
    // Check if the symbol is contained in QuoteBars object
    if (slice.QuoteBars.ContainsKey(_symbol))
    {
        // Obtain the mapped QuoteBar of the symbol
        var quoteBar = slice.QuoteBars[_symbol];
    }
}
```

You can also iterate through the `QuoteBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `QuoteBar` objects.

```
C#
public override void OnData(Slice slice)
{
    // Iterate all received Symbol-QuoteBar key-value pairs
    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        var askPrice = quoteBar.Ask.Close;
    }
}
```

`QuoteBar` objects let LEAN incorporate spread costs into your simulated trade fills to make backtest results more realistic.

`QuoteBar` objects have the following properties:

Ticks

`Tick` objects represent a single trade or quote at a moment in time. A trade tick is a record of a transaction for the Forex pair. A quote tick is an offer to buy or sell the Forex pair at a specific price.

Trade ticks have a non-zero value for the `Quantity` and `Price` properties, but they have a zero value for the `BidPrice` , `BidSize` , `AskPrice` , and `AskSize` properties. Quote ticks have non-zero values for `BidPrice` and `BidSize` properties or have non-zero values for `AskPrice` and `AskSize` properties. To check if a tick is a trade or a quote, use the `TickType` property.

In backtests, LEAN groups ticks into one millisecond buckets. In live trading, LEAN groups ticks into ~70-millisecond buckets. To get the `Tick` objects in the `Slice` , index the `Ticks` property of the `Slice` with a `Symbol` . If the Forex pair doesn't actively trade or you are in the same time step as when you added the Forex pair subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your Forex pair before you index the `Slice` with the Forex pair `Symbol` .

```
C#
public override void OnData(Slice slice)
{
    if (slice.Ticks.ContainsKey(_symbol))
    {
        var ticks = slice.Ticks[_symbol];
        foreach (var tick in ticks)
        {
            var price = tick.Price;
        }
    }
}
```

You can also iterate through the `Ticks` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the

`List<Tick>` objects.

```
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Ticks)
    {
        var symbol = kvp.Key;
        var ticks = kvp.Value;
        foreach (var tick in ticks)
        {
            var price = tick.Price;
        }
    }
}
```

Tick data is raw and unfiltered, so it can contain bad ticks that skew your trade results. For example, some ticks come from dark pools, which aren't tradable. We recommend you only use tick data if you understand the risks and are able to perform your own online tick filtering.

`Tick` objects have the following properties:

Examples

The following examples demonstrate some common practices for handling Forex data.

Example 1: Trade the USDJPY Spread

USDJPY is a Forex pair where the reserve banks of the base currency and quote currency act at different time, which results in a 12-hour period of seasonal peak volume. To reduce slippage costs, place orders when bid-ask spread is relatively low. The following algorithm demonstrates how to utilize the Minimum indicator to check for a narrow spread.

```C#
public class ForexExampleAlgorithm : QCAlgorithm
{
    private dynamic _forex;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add the USDJPY trading pair.
        _forex = AddForex("USDJPY");
        // Create a Minimum indicator to track the lowest bid-ask spread for the past 12 hours.
        _forex.SpreadLow = new Minimum(12*60);
        // Warm up the indicator so it's immediately ready to use.
        WarmUpIndicator<IndicatorDataPoint>(_forex.Symbol, _forex.SpreadLow);
    }

    public override void OnData(Slice slice)
    {
        // Ensure we have quote data for USDJPY in the current slice.
        if (!slice.QuoteBars.ContainsKey(_forex.Symbol))
        {
            return;
        }
        var quoteBar = slice.QuoteBars[_forex.Symbol];
        // Bid-ask spread = Ask price - Bid price
        var bidAskSpread = quoteBar.Ask.Close - quoteBar.Bid.Close;
        // Update the spread minimum indicator to calculate the lowest bid-ask spread over the last 12 hours.
        _forex.SpreadLow.Update(quoteBar.EndTime, bidAskSpread);

        // Trade if the current spread is the lowest bid-ask spread, since it is the most efficient,
        // liquid price with lowest slippage.
        if (!Portfolio.Invested && bidAskSpread == _forex.SpreadLow.Current.Value)
        {
            MarketOrder(_forex.Symbol, 1000);
        }

        // Plot the bid-ask spread to validate the minimum bid-ask spread calculation.
        Plot("Bid-Ask Spread", "spread", bidAskSpread);
    }
}
```

Example 2: 5-Minute EURUSD Trend Reversal

The following algorithm applies a mean reveral strategy to EURUSD. To smooth out the noise of minute resolution data, it consolidates the data into 5-minute bars and then calculates their 200-bar Exponential Moving Average (EMA). After every time the indicator updates, the algorithm checks for a trading opportuniity. When the current EURUSD price is below the EMA, it takes a long position. Otherwise, it takes a short position.

C#

```
public class ForexExampleAlgorithm : QCAlgorithm
{
    private dynamic _forex;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add EURUSD minute data.
        _forex = AddForex("EURUSD");
        // Create an EMA indicator to estimate the trend of EURUSD.
        _forex.ema = new ExponentialMovingAverage(200);
        // Attach an event handler to the indicator to track its updates.
        (_forex.ema as ExponentialMovingAverage).Updated += OnUpdated;
        // Create a consolidator that produces 5-minute quote bars to reduce reduce noise.
        var consolidator = new QuoteBarConsolidator(TimeSpan.FromMinutes(5));
        // Define the consolidator handler so it updates the EMA with the consoildated bars.
        consolidator.DataConsolidated += (_, bar) => _forex.ema.Update(bar.EndTime, bar.Close);
        // Warm up the indicator with historical data.
        foreach (var bar in History<QuoteBar>(_forex.Symbol, 1000))
        {
            consolidator.Update(bar);
        }
        // Subscribe the consolidator for automatic updates.
        SubscriptionManager.AddConsolidator(_forex.Symbol, consolidator);
    }

    private void OnUpdated(object sender, IndicatorDataPoint point)
    {
        // Wait until the indicator is ready and the Forex pair has a price.
        if (!_forex.ema.IsReady || _forex.Price == 0)
        {
            return;
        }
        // Rebalance the portfolio. If the current price is below the EMA, long. Otherwise, short.
        SetHoldings(_forex.Symbol, _forex.Price <= point.Value ? 0.5m : -0.5m);
    }
}
```

Example 3: New York Time Alignment

The following algorithm creates daily Forex bars that close at 4:30 PM Eastern Time (ET). This technique is helpful when you want to align the daily bars with the close of the New York.

C#

```
public class ForexExampleAlgorithm : QCAlgorithm
{
    private Symbol _forex;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);;
        // Add EURUSD minute data.
        _forex = AddForex("EURUSD", Resolution.Minute, Market.Oanda).Symbol;
        // Create a daily bar consolidator with a custom period (4:30 PM-4:30 PM Eastern Time).
        var consolidator = new QuoteBarConsolidator(ConsolidationPeriod);
        // Attach a consolidation handler to plot the consolidated bars.
        consolidator.DataConsolidated += (_, bar) => Plot("Consolidated Bar", "Close", bar.Close);
        // Subscribe the consolidator for automatic updates.
        SubscriptionManager.AddConsolidator(_forex, consolidator);
    }

    private CalendarInfo ConsolidationPeriod(DateTime dt)
    {
        // Set the start time to be 4:30 PM of the previous day.
        var start = dt.Date.AddHours(16).AddMinutes(30);
        var period = TimeSpan.FromDays(1);
        if (dt.Hour < 17)
        {
            start = start - period;
        }
        // Set the period to 1 day to make daily bars.
        return new CalendarInfo(start, period);
    }
}
```

Forex

Market Hours

Introduction

This page shows the trading hours, holidays, and time zone of the Forex market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Forex market:

Weekday	**Time (America/New York)**
Sunday	17:03:00 to 24:00:00
Monday	00:00:00 to 16:58:00, 17:03:00 to 24:00:00
Tuesday	00:00:00 to 16:58:00, 17:03:00 to 24:00:00
Wednesday	00:00:00 to 16:58:00, 17:03:00 to 24:00:00
Thursday	00:00:00 to 16:58:00, 17:03:00 to 24:00:00
Friday	00:00:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Forex market:

Date (*yyyy-mm-dd*)				
2005-01-01	2010-01-01	2010-12-25	2011-01-01	2012-12-25
2013-01-01	2015-12-25	2016-01-01	2016-12-25	2017-01-01
2020-12-25	2021-01-01	2022-12-25	2022-12-26	2023-01-01
2023-01-02	2023-12-24	2023-12-31		

Early Closes

The following table shows the early closes for the Forex market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2005-12-29	17:00:00
2005-12-30	17:00:00
2007-12-31	17:00:00
2008-12-31	17:00:00
2009-12-31	17:00:00
2010-12-31	17:00:00
2011-12-30	17:00:00
2012-12-31	14:00:00
2013-12-31	14:00:00
2014-12-31	14:00:00
2015-12-31	14:00:00
2016-12-30	14:00:00
2017-12-29	17:00:00
2018-12-24	17:00:00
2018-12-31	17:00:00
2019-12-24	17:00:00
2019-12-31	17:00:00
2020-12-24	17:00:00
2020-12-31	17:00:00
2021-12-31	17:00:00
2024-12-24	17:00:00
2024-12-31	17:00:00
2025-12-24	17:00:00
2025-12-31	17:00:00

Late Opens

The following table shows the late opens for the Forex market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2018-12-25	17:00:00
2019-01-01	17:00:00
2019-12-25	17:00:00
2020-01-01	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-12-25	17:05:00
2025-01-01	17:05:00
2025-12-25	17:05:00
2026-01-01	17:05:00

Time Zone

The Forex market trades in the `America/New York` time zone.

Assets With Other Hours

The following list shows the pairs that have different trading periods than the overall Forex market:

- EURNZD
- GBPNZD
- NZDCAD
- NZDCHF
- NZDHKD
- NZDJPY
- NZDSGD
- NZDUSD

Market Hours

EURNZD

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/NZD pair in the Forex market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/NZD pair in the Forex market:

Weekday	**Time (America/New York)**
Sunday	17:03:00 to 24:00:00
Monday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Tuesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Wednesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Thursday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Friday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Forex market:

Date (*yyyy-mm-dd*)				
2005-01-01	2010-01-01	2010-12-25	2011-01-01	2012-12-25
2013-01-01	2015-12-25	2016-01-01	2016-12-25	2017-01-01
2020-12-25	2021-01-01	2022-12-25	2022-12-26	2023-01-01
2023-01-02	2023-12-24	2023-12-31		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR/NZD pair in the Forex market trades in the `America/New York` time zone.

Market Hours

GBPNZD

Introduction

This page shows the trading hours, holidays, and time zone of the GBP/NZD pair in the Forex market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GBP/NZD pair in the Forex market:

Weekday	**Time (America/New York)**
Sunday	17:03:00 to 24:00:00
Monday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Tuesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Wednesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Thursday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Friday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Forex market:

Date (*yyyy-mm-dd*)				
2005-01-01	2010-01-01	2010-12-25	2011-01-01	2012-12-25
2013-01-01	2015-12-25	2016-01-01	2016-12-25	2017-01-01
2020-12-25	2021-01-01	2022-12-25	2022-12-26	2023-01-01
2023-01-02	2023-12-24	2023-12-31		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The GBP/NZD pair in the Forex market trades in the `America/New York` time zone.

Market Hours

NZDCAD

Introduction

This page shows the trading hours, holidays, and time zone of the NZD/CAD pair in the Forex market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NZD/CAD pair in the Forex market:

Weekday	**Time (America/New York)**
Sunday	17:03:00 to 24:00:00
Monday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Tuesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Wednesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Thursday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Friday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Forex market:

Date (*yyyy-mm-dd*)				
2005-01-01	2010-01-01	2010-12-25	2011-01-01	2012-12-25
2013-01-01	2015-12-25	2016-01-01	2016-12-25	2017-01-01
2020-12-25	2021-01-01	2022-12-25	2022-12-26	2023-01-01
2023-01-02	2023-12-24	2023-12-31		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The NZD/CAD pair in the Forex market trades in the `America/New York` time zone.

Market Hours

NZDCHF

Introduction

This page shows the trading hours, holidays, and time zone of the NZD/CHF pair in the Forex market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NZD/CHF pair in the Forex market:

Weekday	**Time (America/New York)**
Sunday	17:03:00 to 24:00:00
Monday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Tuesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Wednesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Thursday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Friday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Forex market:

Date (*yyyy-mm-dd*)				
2005-01-01	2010-01-01	2010-12-25	2011-01-01	2012-12-25
2013-01-01	2015-12-25	2016-01-01	2016-12-25	2017-01-01
2020-12-25	2021-01-01	2022-12-25	2022-12-26	2023-01-01
2023-01-02	2023-12-24	2023-12-31		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The NZD/CHF pair in the Forex market trades in the `America/New York` time zone.

Market Hours

NZDHKD

Introduction

This page shows the trading hours, holidays, and time zone of the NZD/HKD pair in the Forex market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NZD/HKD pair in the Forex market:

Weekday	**Time (America/New York)**
Sunday	17:03:00 to 24:00:00
Monday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Tuesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Wednesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Thursday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Friday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Forex market:

Date (*yyyy-mm-dd*)				
2005-01-01	2010-01-01	2010-12-25	2011-01-01	2012-12-25
2013-01-01	2015-12-25	2016-01-01	2016-12-25	2017-01-01
2020-12-25	2021-01-01	2022-12-25	2022-12-26	2023-01-01
2023-01-02	2023-12-24	2023-12-31		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The NZD/HKD pair in the Forex market trades in the `America/New York` time zone.

Market Hours

NZDJPY

Introduction

This page shows the trading hours, holidays, and time zone of the NZD/JPY pair in the Forex market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NZD/JPY pair in the Forex market:

Weekday	**Time (America/New York)**
Sunday	17:03:00 to 24:00:00
Monday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Tuesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Wednesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Thursday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Friday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Forex market:

Date (*yyyy-mm-dd*)				
2005-01-01	2010-01-01	2010-12-25	2011-01-01	2012-12-25
2013-01-01	2015-12-25	2016-01-01	2016-12-25	2017-01-01
2020-12-25	2021-01-01	2022-12-25	2022-12-26	2023-01-01
2023-01-02	2023-12-24	2023-12-31		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The NZD/JPY pair in the Forex market trades in the `America/New York` time zone.

Market Hours

NZDSGD

Introduction

This page shows the trading hours, holidays, and time zone of the NZD/SGD pair in the Forex market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NZD/SGD pair in the Forex market:

Weekday	**Time (America/New York)**
Sunday	17:03:00 to 24:00:00
Monday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Tuesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Wednesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Thursday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Friday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Forex market:

Date (*yyyy-mm-dd*)				
2005-01-01	2010-01-01	2010-12-25	2011-01-01	2012-12-25
2013-01-01	2015-12-25	2016-01-01	2016-12-25	2017-01-01
2020-12-25	2021-01-01	2022-12-25	2022-12-26	2023-01-01
2023-01-02	2023-12-24	2023-12-31		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The NZD/SGD pair in the Forex market trades in the `America/New York` time zone.

Market Hours

NZDUSD

Introduction

This page shows the trading hours, holidays, and time zone of the NZD/USD pair in the Forex market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NZD/USD pair in the Forex market:

Weekday	**Time (America/New York)**
Sunday	17:03:00 to 24:00:00
Monday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Tuesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Wednesday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Thursday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00, 17:03:00 to 24:00:00
Friday	00:00:00 to 12:58:00, 13:03:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Forex market:

Date (*yyyy-mm-dd*)				
2005-01-01	2010-01-01	2010-12-25	2011-01-01	2012-12-25
2013-01-01	2015-12-25	2016-01-01	2016-12-25	2017-01-01
2020-12-25	2021-01-01	2022-12-25	2022-12-26	2023-01-01
2023-01-02	2023-12-24	2023-12-31		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The NZD/USD pair in the Forex market trades in the `America/New York` time zone.

Asset Classes

Futures

A Futures contract is an agreement to buy or sell a particular commodity, asset, or security at a predetermined price at a specified time in the future.

Requesting Data

Handling Data

Market Hours

See Also

BasicTemplateFuturesAlgorithm.py
BasicTemplateFuturesAlgorithm.cs

Futures

Requesting Data

Request Futures data in your algorithm to receive a feed of contract prices in the `OnData` method. For more information about the specific dataset we use for backtests, see the US Futures dataset listing . To trade Futures live, you can use the QuantConnect data provider or one of the brokerage data providers .

To request Futures data, call one of the following methods:

- `AddFuture` to add a basket of contracts for a single Future
- `AddFutureContract` to add individual contracts for an underlying Future

Request Continuous Contracts

`AddFuture`

Request and Filter All Contracts

`AddFuture`

Select Individual Contracts

`AddFutureContract`

Requesting Data

Universes

Introduction

The `AddFuture` method enables you to select a basket of Future contracts for an underlying Future. To form your universe of contracts, you can filter them down by their expiry. If you want to subscribe to individual contracts one-by-one instead of a set of contracts, see Individual Contracts .

Create Universes

To add a universe of Future contracts, in the `Initialize` method, call the `AddFuture` method. This method returns an `Future` object, which has a `SetFilter` method you can call to filter the set of tradable contract down to just the contracts you want.

```C#
public class BasicFutureAlgorithm : QCAlgorithm
{
    private Future _future;
    public override void Initialize()
    {
        _future = AddFuture(
            Futures.Indices.SP500EMini,
            Resolution.Daily,
            extendedMarketHours: true,
            dataMappingMode: DataMappingMode.OpenInterest,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            contractDepthOffset: 0
        );
        _future.SetFilter(0, 182);
    }

    public override void OnData(Slice data)
    {
        foreach (var (continuousSymbol, chain) in data.FuturesChains)
        {
            foreach (var (symbol, contract) in chain.Contracts)
            {
                var expiry = contract.Expiry;
            }
        }
    }
}
```

For more information about the `AddFuture` method, see Create Universes .

Continous Contracts

By default, LEAN only subscribes to the continuous Future contract. A continuous Future contract represents a series of separate contracts stitched together to form a continuous price. If you need a lot of historical data to warm up an indicator, apply the indicator to the continuous contract price series. The `Future` object has a `Symbol` property and a `Mapped` property. The price of the `Symbol` property is the adjusted price of the continuous contract. The price of the `Mapped` property is the raw price of the currently selected contract in the continuous contract series.

```
// Get the adjusted price of the continuous contract.
var adjustedPrice = Securities[_future.Symbol].Price;

// Get the raw price of the currently selected contract in the continuous contract series.
var rawPrice = Securities[_future.Mapped].Price;
```

The continuous Futures contract isn't a tradable security . You must place orders for a specific Futures contract. To access the currently selected contract in the continuous contract series, use the `Mapped` property of the `Future` object.

```
// Place a market order for the currently selected contract in the continuous contract series.
MarkerOrder(_future.Mapped, 1);
```

For more information, see Continous Contracts .

Filter by Contract Properties

To set a contract filter, in the `Initialize` method, call the `SetFilter` method of the `Future` object. The following table describes the available filter techniques:

`SetFilter(int minExpiry, int maxExpiry)` Selects the contracts that expire within the range you set. This filter runs asynchronously by default.
`SetFilter(Func<FutureFilterUniverse, FutureFilterUniverse> universeFunc)` Selects the contracts that a function selects.

```
// Select the contracts that expire within 182 days.
_future.SetFilter(0, 182);

// Select the front month contract.
_future.SetFilter(futureFilterUniverse => futureFilterUniverse.FrontMonth());
```

The following table describes the filter methods of the `FutureFilterUniverse` class:

`StandardsOnly()` Selects standard contracts
`FrontMonth()` Selects the front month contract
`BackMonths()` Selects the non-front month contracts
`BackMonth()` Selects the back month contracts
`Expiration(TimeSpan minExpiry, TimeSpan maxExpiry)` Selects contracts that expire within a range of dates relative to the current day
`Expiration(int minExpiryDays, int maxExpiryDays)` Selects contracts that expire within a range of dates relative to the current day
`Contracts(IEnumerable<Symbol> contracts)` Selects a list of contracts
`Contracts(Func<IEnumerable<Symbol>, IEnumerable< Symbol>> contractSelector)` Selects contracts that a selector function selects

The preceding methods return an `FutureFilterUniverse` , so you can chain the methods together.

```
// Select the front month standard contracts
_future.SetFilter(futureFilterUniverse => futureFilterUniverse.StandardsOnly().FrontMonth());
```

You can also define an isolated filter method.

```
// In Initialize
_future.SetFilter(Selector);

private FutureFilterUniverse Selector(FutureFilterUniverse futureFilterUniverse)
{
    return futureFilterUniverse.StandardsOnly().FrontMonth();
}
```

Default Filter

By default, LEAN doesn't add any contracts to the FuturesChain it passes to the `OnData` method.

Historical Data

To get historical data for a Futures universe, see Historical Data .

Examples

The following examples demonstrate some common practices for requesting Futures universe data.

Example 1: Rollover

Future contracts expire monthly or quarterly in most cases. Hence, if we hold Future position in month or quarter end, we must consider rolling over to the mapped contract. The following algorithm shows how to buy and roll over to the next front month Future contract. We make use of the universe selection filter to select the front month contract and order the next mapped contract during the previous one expires.

```
C#
public class FutureExampleAlgorithm : QCAlgorithm
{
    private Symbol _future;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Seed the price with last known price to ensure the contract price data is available for the limit order.
        Settings.SeedInitialPrices = true;
        var future = AddFuture(Futures.Indices.SP500EMini, extendedMarketHours: true);
        _future = future.Symbol;
        // We only want to hold position of the front month contract.
        future.SetFilter(u => u.FrontMonth());
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        // Liquidate if expired (or not being the front month contract anymore) and exit universe.
        foreach (var removed in changes.RemovedSecurities)
        {
            Liquidate(removed.Symbol);
        }

        foreach (var added in changes.AddedSecurities)
        {
            // Make sure the newly added contract is an actual mapped tradable contract.
            if (!added.Symbol.IsCanonical())
            {
                // Roll over by ordering the same quantity.
                // Use limit order since market on open order is not supported on Future and avoid extreme quote
filling.
                LimitOrder(added.Symbol, 1m, added.Price);
            }
        }
    }
}
```

Example 2: Continuous Future Indicator

One of the major applications of Continuous Futures is to obtain a smooth price series to feed into indicators. This technique ensures the indicator gets the price data that is comparable to the current contract in the continuous contract series. This example demonstrates a Exponential Moving Average crossover strategy based on the continuous ES contract data.

C#

```
public class FutureEMACrossAlgorithm : QCAlgorithm
{
    private Future _future;
    private decimal _threshold = 0.01m;  // 1% above/below the EMA.

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);
        // Seed the price of each asset with its last known price to
        // avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Use backward ratio normalization for the continuous contract
        // so the indicator receives a smooth price series.
        _future = AddFuture(
            Futures.Indices.SP500EMini,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio
        );
        // Only add the front-month contract.
        _future.SetFilter(u => u.FrontMonth());
        // Create a 252-day EMA indicator as a trend estimator.
        ((dynamic)_future).ema = EMA(_future.Symbol, 252, Resolution.Daily);
        // Warm up the EMA indicator to so we can trade right away.
        WarmUpIndicator(_future.Symbol, _future.Get<ExponentialMovingAverage>("ema"));
    }

    public override void OnData(Slice slice)
    {
        // Ensure the TradeBar data is available for the Future before
        // placing trades.
        if (!slice.Bars.TryGetValue(_future.Symbol, out var bar))
        {
            return;
        }
        // Buy the mapped contract if the trend is estimated to go up
        // (price above the EMA).
        var ema = _future.Get<ExponentialMovingAverage>("ema");
        if (bar.Close >= ema * (1 + _threshold))
        {
            SetHoldings(_future.Mapped, 0.1m);
        }
        // Short the mapped contract if the trend is estimated to go down
        // (price below the EMA).
        else if (bar.Close <= ema * (1 - _threshold))
        {
            SetHoldings(_future.Mapped, -0.1m);
        }
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        // Liquidate the contract when it is no longer the front-month
        // contract (expired).
        foreach (var contract in changes.RemovedSecurities)
        {
            Liquidate(contract.Symbol);
        }
    }
}
```

Example 3: Contango

In Future trading, contango refers to the far-to-expiry Future contract price is higher than the spot price due to various reasons, such as storage fee and insurance of the commodities. The following example shows a contango trading by shorting the far contract that the price is above a threshold compared to the front month contract price and buying the front month contract to earn the premium in between.

C#

```
public class FutureExampleAlgorithm : QCAlgorithm
{
    private Symbol _future;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add Gold Futures.
        var future = AddFuture(Futures.Metals.Gold);
        _future = future.Symbol;
        // Limit the expiration to within 6 months, as the longer the expiration, the higher the price uncertainty.
        future.SetFilter((u) => u.Expiration(0, 183));
    }

    public override void OnData(Slice slice)
    {
        // Get Future chain only for the selected Future contract.
        if (!Portfolio.Invested && slice.FutureChains.TryGetValue(_future, out var chain))
        {
            // It takes 2 contracts with different expiries to form a horizontal spread arbitration to earn price
difference in contango.
            if (chain.Count() < 2) return;
            var farContract = chain.MaxBy(x => x.Expiry);
            var nearContract = chain.MinBy(x => x.Expiry);

            // Check if the far contract price is 1% higher than the near one.
            // If so, short the far contract and buy the near one to earn the horizontal spread premium.
            if (farContract.BidPrice >= nearContract.AskPrice * 1.01m)
            {
                MarketOrder(farContract.Symbol, -1);
                MarketOrder(nearContract.Symbol, 1);
            }
        }
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        // Liquidate if expired (or not being the front month contract anymore) and exit universe.
        foreach (var removed in changes.RemovedSecurities)
        {
            Liquidate();
        }
    }
}
```

Requesting Data

Individual Contracts

Introduction

The `AddFutureContract` method enables you to add an individual Future contract to your algorithm. To check which contracts are currently available to add to your algorithm, call the `FuturesChain` method. If you want to subscribe to a set of contracts instead of individual contracts one-by-one, see Universes .

Create Subscriptions

Before you can subscribe to a Future contract, you must have the underlying Future `Symbol` to get the contract `Symbol` .

```
C#
public class BasicFutureAlgorithm : QCAlgorithm
{
    private Symbol _continuous, _contractSymbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        _continuous = AddFuture(Futures.Indices.SP500EMini).Symbol;
    }

    public override void OnData(Slice data)
    {
        if (_contractSymbol == null)
        {
            _contractSymbol = FuturesChain(_continuous).First().Symbol;
            AddFutureContract(_contractSymbol);
        }
    }
}
```

Get Contract Symbols

To subscribe to a Future contract, you need the contract `Symbol` . The preferred method to getting Future contract `Symbol` objects is to use the `FuturesChain` method. This method returns a `FuturesChain` object, which represent an entire chain of Option contracts for a single underlying security. With the chain, sort and filter the data to find the specific contract(s) you want to trade.

```
C#
// Select the first contract in the chain, which is the front-month contract.
_contractSymbol = FuturesChain(_continuous).First().Symbol;
```

Alternatively, you can create a Future contract `Symbol` object directly parsing its ticker using the `SymbolRepresentation.ParseFutureSymbol` method.

```
C#
// Create the contract Symbol for the December 2025 E-mini S&P 500 Future.
_contractSymbol = SymbolRepresentation.ParseFutureSymbol("ESZ25");
// Create the contract Symbol using the 1-digit year format.
_contractSymbol = SymbolRepresentation.ParseFutureSymbol("ESZ5");
// To avoid ambiguity, specify the 4-digit year.
_contractSymbol = SymbolRepresentation.ParseFutureSymbol("ESZ5", 2015);
```

Subscribe to Contracts

To create an Future contract subscription, pass the contract `Symbol` to the `AddFutureContract` method. Save a reference to the contract `Symbol` so you can easily access the Future contract in the FutureChain that LEAN passes to the `OnData` method. This method returns an `Future` object.

```
C#
AddFutureContract(_contractSymbol);
```

The `AddFutureContract` method creates a subscription for a single Future contract and adds it to your user-defined universe. To create a dynamic universe of Futures contracts, add an Future universe or an Future Universe Selection model .

Warm Up Contract Prices

If you subscribe to an Future contract with `AddFutureContract` , you'll need to wait until the next `Slice` to receive data and trade the contract. To trade the contract in the same time step you subscribe to the contract, set the current price of the contract in a security initializer .

```
C#
Settings.SeedInitialPrices = true;
```

Supported Assets

To view the supported assets in the Futures dataset, see the Supported Assets .

Resolutions

The following table shows the available resolutions and data formats for Futures subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`			✓	✓
`Second`	✓	✓		
`Minute`	✓	✓		
`Hour`	✓	✓		
`Daily`	✓	✓		

The default resolution for Futures contract subscriptions is `Resolution.Minute` . To change the resolution, pass a `resolution` argument to the `AddFutureContract` method.

```
C#
AddFutureContract(_contractSymbol, Resolution.Daily);
```

To create custom resolution periods, see Consolidating Data .

Supported Markets

The following `Market` enumeration members are available for Futures:

Historical data for backtesting is unavailable for `ICE` , `INDIA` , `SGX` and `NYSELIFFE` . In live trading, LEAN sources this data from

your brokerage or a third-party data provider .

You don't need to pass a `market` argument to the `AddFutureContract` method because the contract `Symbol` already contains the market.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
C#
AddFutureContract(_contractSymbol, fillForward: false);
```

Margin and Leverage

LEAN models buying power and margin calls to ensure your algorithm stays within the margin requirements. Futures are already leveraged products, so you can't change their leverage with the default margin model .

Extended Market Hours

By default, your security subscriptions only cover regular trading hours. To subscribe to pre and post-market trading hours for a specific asset, enable the `extendedMarketHours` argument when you create the security subscription.

```
C#
AddFutureContract(_contractSymbol, extendedMarketHours: true);
```

You only receive extended market hours data if you create the subscription with minute, second, or tick resolution. If you create the subscription with daily or hourly resolution, the bars only reflect the regular trading hours.

To view the schedule of regular and extended market hours, see Market Hours .

In general, we model most Futures market hours with the following segments:

Market Segment	**Time**
Pre-market	00:00:00 to 09:30:00
Market	09:30:00 to 17:00:00
Post-market	18:00:00 to 00:00:00

We model it this way because some Futures, like VIX, have pre- and post-market hours, so we standardized it. With this segmentation, if you set a Scheduled Events for the market open, it's set for 9:30 AM instead of midnight.

Continuous Contracts

A continuous Futures contract represents a single contract that maps to other contracts over time as the rollover rules are met. For more information about continuous Futures contracts, see Continuous Contracts .

Data Normalization

The data normalization mode doesn't affect the data that LEAN passes to `OnData` or the data from history request for Futures contracts. If you change the data normalization mode, it won't change the outcome.

The following data normalization modes are available for continuous Futures contracts :

Properties

The `AddFutureContract` method returns a `Future` object, which have the following properties:

Examples

The following examples demonstrate common practices for requesting individual Future contract data.

Example 1: Add the Front-Month Contract

The following algorithm adds the front-month contract, with a two-day buffer, for the E-Mini S&P 500 Future and plots the daily closing price of the contract three days before month end. Then, trade an EMA cross strategy.

C#

```
public class FutureExampleAlgorithm : QCAlgorithm
{
    private Symbol _future, _contractSymbol;
    private ExponentialMovingAverage _ema;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request the future underlying data for ATM strike selection.
        _future = AddFuture(
            Futures.Indices.SP500EMini,
            extendedMarketHours: true,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio
        ).Symbol;

        // Create an EMA indicator for trade signal generation.
        _ema = EMA(_future, 20, Resolution.Daily);
        // Warm up the indicator for immediate readiness to trade with.
        WarmUpIndicator(_future, _ema, Resolution.Daily);

        // Set a scheduled event to select the option contract periodically.
        Schedule.On(
            // 3 days offset to select in prior.
            DateRules.MonthEnd(3),
            TimeRules.At(0, 1),
            SelectContract
        );
        SelectContract();
    }

    private void SelectContract()
    {
        // Get all the contracts that are currently tradable.
        var chain = FuturesChain(_future);
        // Select the front-month contract due to liquidity, with a 2-day buffer to avoid volatility.
        var contracts = chain.Where(contract => contract.Expiry >= Time.AddDays(2));
        if (contracts.Count() == 0)
        {
            return;
        }
        var contract = contracts.First().Symbol;
        // Request the Futures contract for trading.
        _contractSymbol = AddFutureContract(contract).Symbol;
    }

    public override void OnData(Slice slice)
    {
        // Trade based on updated data.
        if (slice.Bars.TryGetValue(_contractSymbol, out var bar) && IsMarketOpen(_contractSymbol))
        {
            // Trade trend-following strategy with EMA cross.
            if (bar.Close > _ema && !Portfolio[_future].IsLong)
            {
                SetHoldings(_contractSymbol, 0.5m);
            }
            else if (bar.Close < _ema && !Portfolio[_future].IsShort)
            {
                SetHoldings(_contractSymbol, -0.5m);
            }
        }
    }
}
```

Futures

Handling Data

Introduction

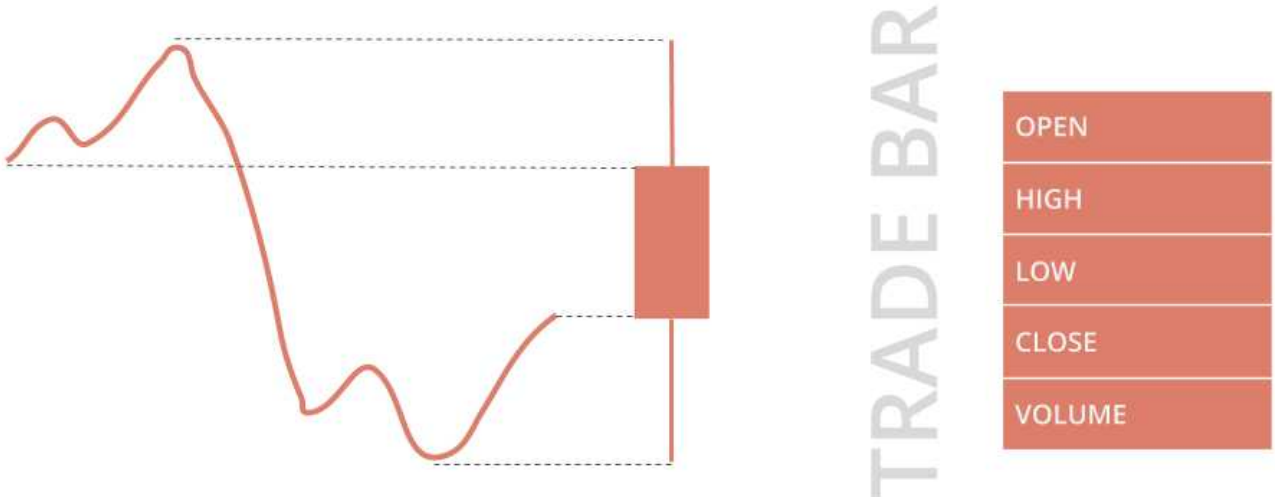
LEAN passes the data you request to the `OnData` method so you can make trading decisions. The default `OnData` method accepts a `Slice` object, but you can define additional `OnData` methods that accept different data types. For example, if you define an `OnData` method that accepts a `TradeBar` argument, it only receives `TradeBar` objects. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `TradeBars` `DataDictionary` is made up of `TradeBar` objects. To access individual data points in the dictionary, you can index the dictionary with the contract ticker or `Symbol` , but we recommend you use the `Symbol` .

To view the resolutions that are available for Futures data, see Resolutions .

Trades

`TradeBar` objects are price bars that consolidate individual trades from the exchanges. They contain the open, high, low, close, and volume of trading activity over a period of time.



To get the `TradeBar` objects in the `Slice` , index the `Slice` or index the `Bars` property of the `Slice` with the contract `Symbol` . If the contract doesn't actively trade or you are in the same time step as when you added the contract subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your contract before you index the `Slice` with the contract `Symbol` .

```C#
public override void OnData(Slice slice)
{
    // Check if the symbol is contained in TradeBars object
    if (slice.Bars.ContainsKey(_contractSymbol))
    {
        // Obtain the mapped TradeBar of the symbol
        var tradeBar = slice.Bars[_contractSymbol];
    }
}
```

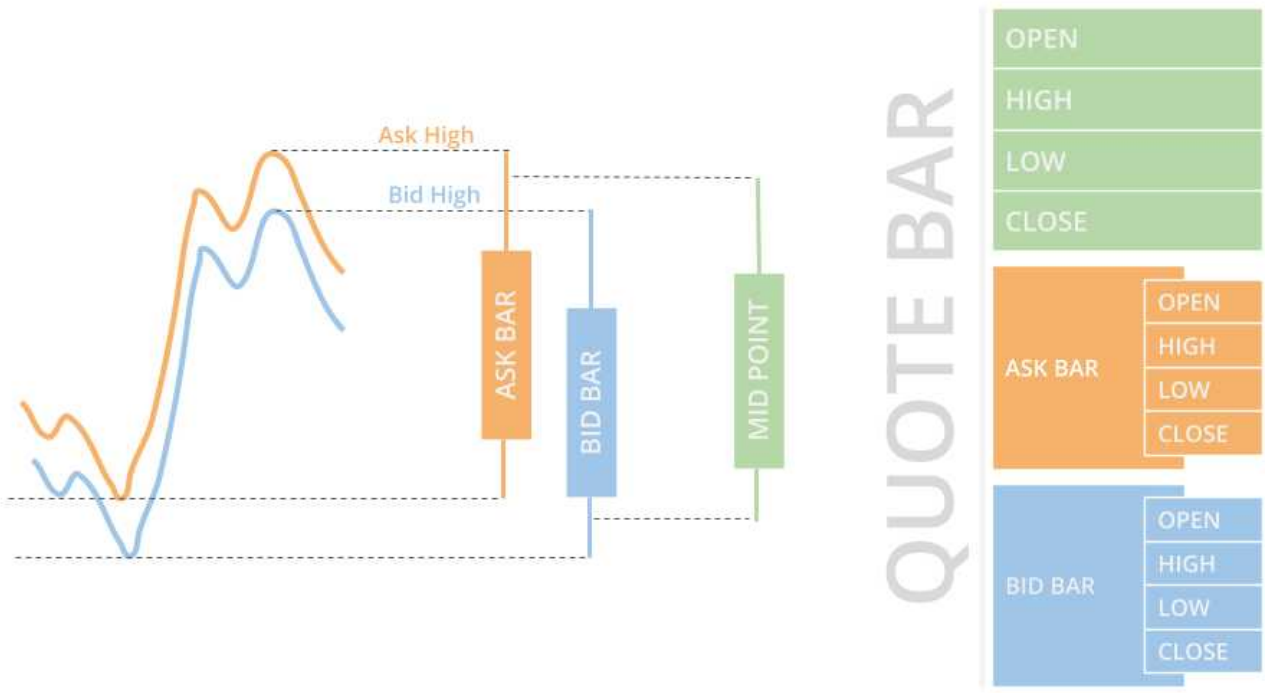
You can also iterate through the `TradeBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `TradeBar` objects.

```C#
public override void OnData(Slice slice)
{
    // Iterate all received Symbol-TradeBar key-value pairs
    foreach (var kvp in slice.Bars)
    {
        var symbol = kvp.Key;
        var tradeBar = kvp.Value;
        var closePrice = tradeBar.Close;
    }
}
```

`TradeBar` objects have the following properties:

Quotes

`QuoteBar` objects are bars that consolidate NBBO quotes from the exchanges. They contain the open, high, low, and close prices of the bid and ask. The `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` object are the mean of the respective bid and ask prices. If the bid or ask portion of the `QuoteBar` has no data, the `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` copy the values of either the `Bid` or `Ask` instead of taking their mean.



To get the `QuoteBar` objects in the `Slice` , index the `QuoteBars` property of the `Slice` with the contract `Symbol` . If the contract doesn't actively get quotes or you are in the same time step as when you added the contract subscription, the `Slice` may not

contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your contract before you index the `Slice` with the contract `Symbol` .

```
C#
public override void OnData(Slice slice)
{
    // Check if the symbol is contained in QuoteBars object
    if (slice.QuoteBars.ContainsKey(_contractSymbol))
    {
        // Obtain the mapped QuoteBar of the symbol
        var quoteBar = slice.QuoteBars[_contractSymbol];
    }
}
```

You can also iterate through the `QuoteBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `QuoteBar` objects.

```
C#
public override void OnData(Slice slice)
{
    // Iterate all received Symbol-QuoteBar key-value pairs
    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        var askPrice = quoteBar.Ask.Close;
    }
}
```

`QuoteBar` objects let LEAN incorporate spread costs into your simulated trade fills to make backtest results more realistic.

`QuoteBar` objects have the following properties:

Ticks

`Tick` objects represent a single trade or quote at a moment in time. A trade tick is a record of a transaction for the contract. A quote tick is an offer to buy or sell the contract at a specific price.

Trade ticks have a non-zero value for the `Quantity` and `Price` properties, but they have a zero value for the `BidPrice` , `BidSize` , `AskPrice` , and `AskSize` properties. Quote ticks have non-zero values for `BidPrice` and `BidSize` properties or have non-zero values for `AskPrice` and `AskSize` properties. To check if a tick is a trade or a quote, use the `TickType` property.

In backtests, LEAN groups ticks into one millisecond buckets. In live trading, LEAN groups ticks into ~70-millisecond buckets. To get the `Tick` objects in the `Slice` , index the `Ticks` property of the `Slice` with a `Symbol` . If the contract doesn't actively trade or you are in the same time step as when you added the contract subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your contract before you index the `Slice` with the contract `Symbol` .

```
C#
public override void OnData(Slice slice)
{
    if (slice.Ticks.ContainsKey(_contractSymbol))
    {
        var ticks = slice.Ticks[_contractSymbol];
        foreach (var tick in ticks)
        {
            var price = tick.Price;
        }
    }
}
```

You can also iterate through the `Ticks` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the

`List<Tick>` objects.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Ticks)
    {
        var symbol = kvp.Key;
        var ticks = kvp.Value;
        foreach (var tick in ticks)
        {
            var price = tick.Price;
        }
    }
}
```

Tick data is raw and unfiltered, so it can contain bad ticks that skew your trade results. For example, some ticks come from dark pools, which aren't tradable. We recommend you only use tick data if you understand the risks and are able to perform your own online tick filtering.

`Tick` objects have the following properties:

Futures Chains

`FuturesChain` objects represent an entire chain of contracts for a single underlying Future. To get the `FuturesChain` , index the `FuturesChains` property of the `Slice` with the continuous contract `Symbol` .

```
C#
public override void OnData(Slice slice)
{
    // Try to get the FutureChain using the canonical symbol
    if (slice.FuturesChains.TryGetValue(_contractSymbol.Canonical, out var chain))
    {
        // Get all contracts if the FutureChain contains any member
        var contracts = chain.Contracts;
    }
}
```

You can also loop through the `FuturesChains` property to get each `FuturesChain` .

```
C#
public override void OnData(Slice slice)
{
    // Iterate all received Canonical Symbol-FutureChain key-value pairs
    foreach (var kvp in slice.FuturesChains)
    {
        var continuousContractSymbol = kvp.Key;
        var chain = kvp.Value;
        var contracts = chain.Contracts;
    }
}

// Using this overload will only handle any FutureChains object received
public void OnData(FuturesChains futuresChains)
{
    // Iterate all received Canonical Symbol-FutureChain key-value pairs
    foreach (var kvp in futuresChains)
    {
        var continuousContractSymbol = kvp.Key;
        var chain = kvp.Value;
        var contracts = chain.Contracts;
    }
}
```

`FuturesChain` objects have the following properties:

Futures Contracts

`FuturesContract` objects represent the data of a single Futures contract in the market. To get the Futures contracts in the `Slice` , use the `Contracts` property of the `FuturesChain` .

```
C#
public override void OnData(Slice slice)
{
    // Try to get the FutureChain using the canonical symbol
    if (slice.FuturesChains.TryGetValue(_contractSymbol.Canonical, out var chain))
    {
        // Get individual contract data
        if (chain.Contracts.TryGetValue(_contractSymbol, out var contract))
        {
            var price = contract.LastPrice;
        }
    }
}

// // Using this overload will only handle any FutureChains object received
public void OnData(FuturesChains futuresChains)
{
    // Try to get the FutureChain using the canonical symbol
    if (futuresChains.TryGetValue(_contractSymbol.Canonical, out var chain))
    {
        // Get individual contract data
        if (chain.Contracts.TryGetValue(_contractSymbol, out var contract))
        {
            var price = contract.LastPrice;
        }
    }
}
```

Open interest is the number of outstanding contracts that haven't been settled. It provides a measure of investor interest and the market liquidity, so it's a popular metric to use for contract selection. Open interest is calculated once per day. To get the latest open interest value, use the `OpenInterest` property of the `Future` or `FutureContract` .

```
C#
public override void OnData(Slice slice)
{
    // Try to get the FuturesChains using the canonical symbol
    if (slice.FuturesChains.TryGetValue(_contractSymbol.Canonical, out var chain))
    {
        // Get individual contract data
        if (chain.Contracts.TryGetValue(_contractSymbol, out var contract))
        {
            // Get the open interest of the selected contracts
            var openInterest = contract.OpenInterest;
        }
    }
}
```

`FuturesContract` objects have the following properties:

Symbol Changes

When the continuous contract rolls over, LEAN passes a `SymbolChangedEvent` to your `OnData` method, which contains the old contract `Symbol` and the new contract `Symbol` . To get the `SymbolChangedEvent` , use the `SymbolChangedEvents` property of the `Slice` . You can use the `SymbolChangedEvent` to roll over contracts.

```C#
public override void OnData(Slice slice)
{
    foreach (var (symbol, changedEvent) in slice.SymbolChangedEvents)
    {
        var oldSymbol = changedEvent.OldSymbol;
        var newSymbol = changedEvent.NewSymbol;
        var tag = $"Rollover - Symbol changed at {Time}: {oldSymbol} -> {newSymbol}";
        var quantity = Portfolio[oldSymbol].Quantity;
        // Rolling over: to liquidate any position of the old mapped contract and switch to the newly mapped contract
        Liquidate(symbol: oldSymbol, tag: tag);
        if (quantity != 0) MarketOrder(newSymbol, quantity, tag: tag);
        Log(tag);
    }
}
```

LEAN passes a `SymbolChangedEvent` to your `OnSymbolChangedEvents` method as well. LEAN only invokes this method when the continuous contract rolls over.

```C#
public override void OnSymbolChangedEvents(SymbolChangedEvents symbolChangedEvents)
{
    foreach (var (symbol, changedEvent) in symbolChangedEvents)
    {
        var oldSymbol = changedEvent.OldSymbol;
        var newSymbol = changedEvent.NewSymbol;
        var tag = $"Rollover - Symbol changed at {Time}: {oldSymbol} -> {newSymbol}";
        var quantity = Portfolio[oldSymbol].Quantity;
        // Rolling over: to liquidate any position of the old mapped contract and switch to the newly mapped contract
        Liquidate(symbol: oldSymbol, tag: tag);
        if (quantity != 0) MarketOrder(newSymbol, quantity, tag: tag);
        Log(tag);
    }
}
```

In backtesting, the `SymbolChangedEvent` occurs at midnight Eastern Time (ET). In live trading, the live data for continuous contract mapping arrives at 6/7 AM ET, so that's when it occurs.

`SymbolChangedEvent` objects have the following properties:

Examples

The following examples demonstrate some common practices for handling Futures data.

Example 1: Rollovers

Spot Future is referred to as the continuous Future contract, which is usually mapped by the front month contract or the contract with the most open interest. When a contract expires or is very close to expiring, traders usually rollover from the current contract to the next contract to avoid price settlement and remain invested. The following algorithm demonstrates rolling over with limit orders .

C#

```
public class FutureExampleAlgorithm : QCAlgorithm
{
    private Future _future;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add the E-mini Futures and set the continuous contract mapping criteria for the rollovers.
        _future = AddFuture(
            Futures.Indices.SP500EMini,
            extendedMarketHours: true,
            dataMappingMode: DataMappingMode.OpenInterest,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            contractDepthOffset: 0
        );
        _future.SetFilter(0, 182);
    }

    public override void OnData(Slice slice)
    {
        // Place the initial order so you can start rolling over contracts later.
        if (!Portfolio.Invested && Transactions.GetOpenOrders().Count == 0)
        {
            // Buy the contract that's currently selected in the continous contract series.
            MarketOrder(_future.Mapped, 1m);
        }
    }

    // Track rollover events.
    public override void OnSymbolChangedEvents(SymbolChangedEvents symbolChangedEvents)
    {
        foreach (var (symbol, changedEvent) in symbolChangedEvents)
        {
            var oldSymbol = changedEvent.OldSymbol;
            var newSymbol = changedEvent.NewSymbol;
            // The quantity to roll over should be consistent.
            var quantity = Portfolio[oldSymbol].Quantity;
            // Rolling over: Liquidate the old mapped contract and switch to the newly mapped contract.
            var tag = $"Rollover: {oldSymbol} -> {newSymbol}";
            Liquidate(oldSymbol, tag: tag);
            if (quantity != 0)
            {
                // Place a limit order to avoid extreme quote filling.
                var newContract = Securities[newSymbol];
                LimitOrder(
                    newSymbol, quantity,
                    // To avoid warnings, round the target limit price to a price that respects
                    // the minimum price variation for the Future.
                    GetLimitPrice(newContract, newContract.Price),
                    tag: tag
                );
            }
        }
    }

    private decimal GetLimitPrice(Security security, decimal targetLimitPrice, bool roundUp = true)
    {
        var parameters = new GetMinimumPriceVariationParameters(security, targetLimitPrice);
        var pip = security.PriceVariationModel.GetMinimumPriceVariation(parameters);
        return ((int)(targetLimitPrice / pip) + (roundUp ? 1 : 0)) * pip;
    }
}
```

Example 2: Micro Gold vs Gold

The following example demonstrates how to use the `SymbolPropertiesDatabase` to obtain the contract multipliers of both Micro Gold and Gold Future contracts. The algorithm uses it to invest $500k of nominal value into each contract.

C#

```
public class FutureNominalBuysAlgorithm : QCAlgorithm
{
    private List<Future> _futures = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1_000_000);
        // Set the continuous contract mapping criteria for both Futures
        // since we want to trade the most liquid contracts.
        foreach (var ticker in new[] { Futures.Metals.Gold, Futures.Metals.MicroGold })
        {
            var future = AddFuture(
                ticker,
                extendedMarketHours: true,
                dataMappingMode: DataMappingMode.OpenInterest,
                dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
                contractDepthOffset: 0
            );
            future.SetFilter(0, 180);
            _futures.Add(future);
        }
    }

    public override void OnData(Slice slice)
    {
        // Wait until we aren't invested and we have bars for all Futures.
        if (!Portfolio.Invested && _futures.All(f => slice.Bars.ContainsKey(f.Symbol)))
        {
            foreach (var future in _futures)
            {
                // Calculate the order size for $500k. Get the quotient
                // after dividing by the contract multiplier  since the
                // order size must be whole number.
                var quantity = (int)Math.Floor(
                    500_000m / (future.Price * future.SymbolProperties.ContractMultiplier)
                );
                if (quantity == 0) {
                    continue;
                }
                MarketOrder(future.Mapped, quantity);
            }
        }
    }
}
```

Futures

Market Hours

Select one of the following Futures markets to see its operating hours:

CBOT

CFE

CME

COMEX

EUREX

HKFE

ICE

INDIA

NYMEX

NYSELIFFE

SGX

Market Hours

CBOT

Introduction

This page shows the trading hours, holidays, and time zone of the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the CBOT Future market:

Weekday	**Time (America/Chicago)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04

2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-05-29	2023-06-19	2023-07-04

2023-11-23	2023-12-25	2024-01-01	2024-01-15	2024-02-19
2024-03-29	2024-05-27	2024-06-19	2024-07-04	2024-09-02
2024-11-28	2024-12-25	2025-01-09	2025-01-20	2025-04-18

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CBOT Future market trades in the following time zones:

- `America/Chicago`
- `America/New York`

Assets With Other Hours

The following table shows the contracts that have different trading periods than the overall CBOT Future market:

Symbol	Name
10Y	Micro 10-Year Yield Futures
2YY	Micro 2-Year Yield Futures
30Y	Micro 30-Year Yield Futures
5YY	Micro 5-Year Yield Futures
AW	Bloomberg Commodity Index Futures
BCF	Black Sea Corn Financially Settled (Platts) Futures
BWF	Black Sea Wheat Financially Settled (Platts) Futures
EH	Ethanol Futures
F1U	5-Year USD MAC Swap Futures
KE	KC HRW Wheat Futures
MYM	Micro E-mini Dow Jones Industrial Average Index Futures
TN	Ultra 10-Year U.S. Treasury Note Futures
UB	Ultra U.S. Treasury Bond Futures
YM	E-mini Dow ($5) Futures
ZB	U.S. Treasury Bond Futures
ZC	Corn Futures
ZF	5-Year T-Note Futures
ZL	Soybean Oil Futures
ZM	Soybean Meal Futures
ZN	10-Year T-Note Futures
ZO	Oats Futures
ZS	Soybean Futures
ZT	2-Year T-Note Futures
ZW	Chicago SRW Wheat Futures

CBOT

10Y

Introduction

This page shows the trading hours, holidays, and time zone of the Micro 10-Year Yield Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro 10-Year Yield Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro 10-Year Yield Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro 10-Year Yield Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro 10-Year Yield Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro 10-Year Yield Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	10:30:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00
2023-01-16	12:00:00
2023-02-20	12:00:00
2023-04-07	10:15:00
2023-05-29	12:00:00
2023-06-19	12:00:00
2023-07-04	12:00:00
2023-09-04	12:00:00
2023-11-23	12:00:00
2023-11-24	12:15:00
2024-01-15	12:00:00

2024-02-19	12:00:00
2024-03-28	16:00:00
2024-05-27	12:00:00
2024-06-19	12:00:00
2024-07-04	12:00:00
2024-09-02	12:00:00
2024-11-28	12:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	12:00:00
2025-02-17	12:00:00
2025-04-17	16:00:00
2025-05-26	12:00:00
2025-06-19	12:00:00
2025-07-04	12:00:00
2025-09-01	12:00:00
2025-11-27	12:00:00
2025-11-28	12:15:00
2025-12-24	12:15:00
2025-12-25	16:00:00
2025-12-31	16:00:00
2026-01-19	12:00:00
2026-02-16	12:00:00
2026-04-03	10:15:00
2026-05-25	12:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	12:00:00

2026-11-26	12:00:00
2026-11-27	12:15:00
2026-12-24	12:15:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Micro 10-Year Yield Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00

2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00

2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-07-03	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00

2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00

2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Micro 10-Year Yield Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

2YY

Introduction

This page shows the trading hours, holidays, and time zone of the Micro 2-Year Yield Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro 2-Year Yield Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro 2-Year Yield Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro 2-Year Yield Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro 2-Year Yield Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro 2-Year Yield Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	10:30:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00
2023-01-16	12:00:00
2023-02-20	12:00:00
2023-04-07	10:15:00
2023-05-29	12:00:00
2023-06-19	12:00:00
2023-07-04	12:00:00
2023-09-04	12:00:00
2023-11-23	12:00:00
2023-11-24	12:15:00
2024-01-15	12:00:00

2024-02-19	12:00:00
2024-03-28	16:00:00
2024-05-27	12:00:00
2024-06-19	12:00:00
2024-07-04	12:00:00
2024-09-02	12:00:00
2024-11-28	12:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	12:00:00
2025-02-17	12:00:00
2025-04-17	16:00:00
2025-05-26	12:00:00
2025-06-19	12:00:00
2025-07-04	12:00:00
2025-09-01	12:00:00
2025-11-27	12:00:00
2025-11-28	12:15:00
2025-12-24	12:15:00
2025-12-25	16:00:00
2025-12-31	16:00:00
2026-01-19	12:00:00
2026-02-16	12:00:00
2026-04-03	10:15:00
2026-05-25	12:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	12:00:00

2026-11-26	12:00:00
2026-11-27	12:15:00
2026-12-24	12:15:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Micro 2-Year Yield Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00

2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00

2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-07-03	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00

2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00

2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Micro 2-Year Yield Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

30Y

Introduction

This page shows the trading hours, holidays, and time zone of the Micro 30-Year Yield Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro 30-Year Yield Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro 30-Year Yield Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro 30-Year Yield Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro 30-Year Yield Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro 30-Year Yield Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	10:30:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00
2023-01-16	12:00:00
2023-02-20	12:00:00
2023-04-07	10:15:00
2023-05-29	12:00:00
2023-06-19	12:00:00
2023-07-04	12:00:00
2023-09-04	12:00:00
2023-11-23	12:00:00
2023-11-24	12:15:00
2024-01-15	12:00:00

2024-02-19	12:00:00
2024-03-28	16:00:00
2024-05-27	12:00:00
2024-06-19	12:00:00
2024-07-04	12:00:00
2024-09-02	12:00:00
2024-11-28	12:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	12:00:00
2025-02-17	12:00:00
2025-04-17	16:00:00
2025-05-26	12:00:00
2025-06-19	12:00:00
2025-07-04	12:00:00
2025-09-01	12:00:00
2025-11-27	12:00:00
2025-11-28	12:15:00
2025-12-24	12:15:00
2025-12-25	16:00:00
2025-12-31	16:00:00
2026-01-19	12:00:00
2026-02-16	12:00:00
2026-04-03	10:15:00
2026-05-25	12:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	12:00:00

2026-11-26	12:00:00
2026-11-27	12:15:00
2026-12-24	12:15:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Micro 30-Year Yield Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00

2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00

2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-07-03	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00

2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00

2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Micro 30-Year Yield Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

5YY

Introduction

This page shows the trading hours, holidays, and time zone of the Micro 5-Year Yield Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro 5-Year Yield Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro 5-Year Yield Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro 5-Year Yield Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro 5-Year Yield Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro 5-Year Yield Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	10:30:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00
2023-01-16	12:00:00
2023-02-20	12:00:00
2023-04-07	10:15:00
2023-05-29	12:00:00
2023-06-19	12:00:00
2023-07-04	12:00:00
2023-09-04	12:00:00
2023-11-23	12:00:00
2023-11-24	12:15:00
2024-01-15	12:00:00

2024-02-19	12:00:00
2024-03-28	16:00:00
2024-05-27	12:00:00
2024-06-19	12:00:00
2024-07-04	12:00:00
2024-09-02	12:00:00
2024-11-28	12:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	12:00:00
2025-02-17	12:00:00
2025-04-17	16:00:00
2025-05-26	12:00:00
2025-06-19	12:00:00
2025-07-04	12:00:00
2025-09-01	12:00:00
2025-11-27	12:00:00
2025-11-28	12:15:00
2025-12-24	12:15:00
2025-12-25	16:00:00
2025-12-31	16:00:00
2026-01-19	12:00:00
2026-02-16	12:00:00
2026-04-03	10:15:00
2026-05-25	12:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	12:00:00

2026-11-26	12:00:00
2026-11-27	12:15:00
2026-12-24	12:15:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Micro 5-Year Yield Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00

2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00

2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-07-03	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00

2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00

2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Micro 5-Year Yield Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

AW

Introduction

This page shows the trading hours, holidays, and time zone of the Bloomberg Commodity Index Futures contract in the CBOT Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Bloomberg Commodity Index Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:15:00 to 13:30:00
Tuesday	08:15:00 to 13:30:00
Wednesday	08:15:00 to 13:30:00
Thursday	08:15:00 to 13:30:00
Friday	08:15:00 to 13:30:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Bloomberg Commodity Index Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd* **)**				
2023-04-07	2026-01-01	2026-01-19	2026-02-16	2026-04-03
2026-05-25	2026-06-19	2026-07-03	2026-09-07	2026-11-26
2026-12-25	2027-01-01			

Early Closes

The following table shows the early closes for the Bloomberg Commodity Index Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-11-27	12:00:00
2009-12-24	12:00:00
2010-11-26	12:00:00
2011-11-25	12:00:00
2011-12-31	12:00:00
2012-11-23	12:00:00
2012-12-24	12:00:00
2013-11-29	12:00:00
2013-12-24	12:00:00
2014-11-28	12:00:00
2014-12-24	12:00:00
2015-11-27	12:00:00
2015-12-24	12:00:00
2016-11-25	12:00:00
2016-12-23	12:00:00
2017-07-03	12:00:00
2017-11-24	12:00:00
2017-12-22	12:00:00
2018-07-03	12:00:00
2018-11-23	12:00:00
2018-12-24	12:00:00
2019-07-03	12:00:00
2019-11-29	12:00:00
2019-12-24	12:00:00
2026-11-27	12:00:00
2026-12-24	12:00:00

Late Opens

There are no days with late opens.

Time Zone

The Bloomberg Commodity Index Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

BCF

Introduction

This page shows the trading hours, holidays, and time zone of the Black Sea Corn Financially Settled (Platts) Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Black Sea Corn Financially Settled (Platts) Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Black Sea Corn Financially Settled (Platts) Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 13:20:00
Tuesday	08:30:00 to 13:20:00
Wednesday	08:30:00 to 13:20:00
Thursday	08:30:00 to 13:20:00
Friday	08:30:00 to 13:20:00

Post-market Hours

The following table shows the post-market hours for the Black Sea Corn Financially Settled (Platts) Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	19:00:00 to 24:00:00
Tuesday	19:00:00 to 24:00:00
Wednesday	19:00:00 to 24:00:00
Thursday	19:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Black Sea Corn Financially Settled (Platts) Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)				
2023-04-07	2024-01-02	2025-01-01	2025-07-04	2025-11-27
2025-12-25	2026-01-01	2026-04-03	2026-06-19	2026-07-03
2026-11-26	2026-12-25	2027-01-01		

Early Closes

The following table shows the early closes for the Black Sea Corn Financially Settled (Platts) Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-25	12:00:00
2012-01-15	19:00:00
2012-02-19	19:00:00
2012-11-21	16:45:00
2012-11-23	12:00:00
2012-12-24	12:00:00
2013-01-20	16:00:00
2013-02-17	16:00:00
2013-05-26	16:00:00
2013-07-03	12:00:00
2013-09-01	16:00:00
2013-11-27	16:45:00
2013-11-29	12:00:00

2013-12-24	12:00:00
2014-01-19	16:00:00
2014-02-16	16:00:00
2014-05-25	16:00:00
2014-07-03	12:00:00
2014-08-31	16:00:00
2014-11-26	16:45:00
2014-11-28	12:00:00
2014-12-24	12:00:00
2015-01-18	16:00:00
2015-02-15	16:00:00
2015-05-24	16:00:00
2015-07-02	12:00:00
2015-09-06	16:00:00
2015-11-25	16:45:00
2015-11-27	12:05:00
2015-12-24	12:05:00
2016-01-17	16:00:00
2016-02-14	16:00:00
2016-05-29	16:00:00
2016-09-04	16:00:00
2016-11-23	16:45:00
2016-11-25	12:05:00
2016-12-23	12:05:00
2017-01-15	16:00:00
2017-02-19	16:00:00
2017-05-28	16:00:00
2017-07-03	12:05:00

2017-09-03	16:00:00
2017-11-22	16:45:00
2017-12-22	12:05:00
2018-01-14	16:00:00
2018-02-18	16:00:00
2018-05-27	16:00:00
2018-07-03	12:05:00
2018-09-02	16:00:00
2018-11-21	16:45:00
2018-11-23	12:05:00
2018-12-24	12:05:00
2019-01-20	16:00:00
2019-02-17	16:00:00
2019-05-26	16:00:00
2019-07-03	12:05:00
2019-09-01	16:00:00
2019-11-27	16:45:00
2019-11-29	12:05:00
2019-12-24	12:05:00
2020-01-19	16:00:00
2020-02-16	16:00:00
2020-05-24	16:00:00
2020-07-02	12:05:00
2020-09-06	16:00:00
2020-11-25	16:45:00
2020-11-27	12:05:00
2020-12-24	12:05:00
2021-01-17	16:00:00

2021-02-14	16:00:00
2021-05-30	16:00:00
2021-09-05	16:00:00
2021-11-24	16:45:00
2021-11-26	12:05:00
2022-01-16	16:00:00
2022-02-20	16:00:00
2022-05-29	16:00:00
2022-06-19	16:00:00
2022-09-04	16:00:00
2022-11-23	16:45:00
2022-11-25	12:05:00
2023-01-15	16:00:00
2023-02-19	16:00:00
2023-05-28	16:00:00
2023-06-18	16:00:00
2023-09-03	16:00:00
2023-11-22	16:00:00
2023-11-24	12:05:00
2024-03-28	16:00:00
2024-12-24	12:05:00
2024-12-31	16:00:00
2025-01-19	16:00:00
2025-02-16	16:00:00
2025-04-17	16:00:00
2025-05-25	16:00:00
2025-06-18	16:45:00
2025-07-03	16:00:00

2025-08-31	16:00:00
2025-11-26	16:00:00
2025-11-28	12:05:00
2025-12-24	12:05:00
2025-12-31	16:00:00
2026-01-18	16:00:00
2026-02-15	16:00:00
2026-04-02	16:00:00
2026-05-24	16:00:00
2026-06-18	16:00:00
2026-07-02	16:00:00
2026-09-06	16:00:00
2026-11-25	16:00:00
2026-11-27	12:05:00
2026-12-24	12:05:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Black Sea Corn Financially Settled (Platts) Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2011-12-27	09:30:00
2012-01-03	09:30:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-11-23	09:30:00
2013-01-02	09:30:00
2013-01-21	19:00:00
2013-02-18	19:00:00

2013-05-27	19:00:00
2013-07-05	08:30:00
2013-09-02	19:00:00
2013-11-29	08:30:00
2013-12-26	08:30:00
2014-01-02	08:30:00
2014-01-20	19:00:00
2014-02-17	19:00:00
2014-05-26	19:00:00
2014-09-01	19:00:00
2015-01-19	19:00:00
2015-02-16	19:00:00
2015-05-25	19:00:00
2015-09-07	19:00:00
2016-01-18	19:00:00
2016-02-15	19:00:00
2016-05-30	19:00:00
2016-09-05	19:00:00
2016-11-25	08:30:00
2017-01-16	19:00:00
2017-02-20	19:00:00
2017-05-29	19:00:00
2017-07-05	08:30:00
2017-09-04	19:00:00
2017-11-24	08:30:00
2018-01-15	19:00:00
2018-02-19	19:00:00
2018-05-28	19:00:00

2018-07-05	08:30:00
2018-09-03	19:00:00
2018-11-23	08:30:00
2019-01-21	19:00:00
2019-02-18	19:00:00
2019-05-27	19:00:00
2019-07-05	08:30:00
2019-09-02	19:00:00
2019-11-29	08:30:00
2020-01-20	19:00:00
2020-02-17	19:00:00
2020-05-25	19:00:00
2020-09-07	19:00:00
2020-11-27	08:30:00
2021-01-18	19:00:00
2021-02-15	19:00:00
2021-05-31	19:00:00
2021-09-06	19:00:00
2021-11-26	08:30:00
2022-01-17	19:00:00
2022-02-21	19:00:00
2022-05-30	19:00:00
2022-06-20	19:00:00
2022-09-05	19:00:00
2023-01-16	19:00:00
2023-02-20	19:00:00
2023-05-29	19:00:00
2023-06-19	19:00:00

2023-09-04	19:00:00
2024-01-15	19:00:00
2024-02-19	19:00:00
2024-05-27	19:00:00
2024-06-19	19:00:00
2024-09-02	19:00:00
2025-01-20	19:00:00
2025-02-17	19:00:00
2025-05-26	19:00:00
2025-06-19	19:00:00
2025-09-01	19:00:00
2026-01-02	08:30:00
2026-01-19	19:00:00
2026-02-16	19:00:00
2026-05-25	19:00:00
2026-09-07	19:00:00
2026-11-27	08:30:00

Time Zone

The Black Sea Corn Financially Settled (Platts) Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

BWF

Introduction

This page shows the trading hours, holidays, and time zone of the Black Sea Wheat Financially Settled (Platts) Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Black Sea Wheat Financially Settled (Platts) Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Black Sea Wheat Financially Settled (Platts) Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 13:20:00
Tuesday	08:30:00 to 13:20:00
Wednesday	08:30:00 to 13:20:00
Thursday	08:30:00 to 13:20:00
Friday	08:30:00 to 13:20:00

Post-market Hours

The following table shows the post-market hours for the Black Sea Wheat Financially Settled (Platts) Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	19:00:00 to 24:00:00
Tuesday	19:00:00 to 24:00:00
Wednesday	19:00:00 to 24:00:00
Thursday	19:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Black Sea Wheat Financially Settled (Platts) Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)				
2023-04-07	2024-01-02	2025-01-01	2025-07-04	2025-11-27
2025-12-25	2026-01-01	2026-04-03	2026-06-19	2026-07-03
2026-11-26	2026-12-25	2027-01-01		

Early Closes

The following table shows the early closes for the Black Sea Wheat Financially Settled (Platts) Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-25	12:00:00
2012-01-15	19:00:00
2012-02-19	19:00:00
2012-11-21	16:45:00
2012-11-23	12:00:00
2012-12-24	12:00:00
2013-01-20	16:00:00
2013-02-17	16:00:00
2013-05-26	16:00:00
2013-07-03	12:00:00
2013-09-01	16:00:00
2013-11-27	16:45:00
2013-11-29	12:00:00

2013-12-24	12:00:00
2014-01-19	16:00:00
2014-02-16	16:00:00
2014-05-25	16:00:00
2014-07-03	12:00:00
2014-08-31	16:00:00
2014-11-26	16:45:00
2014-11-28	12:00:00
2014-12-24	12:00:00
2015-01-18	16:00:00
2015-02-15	16:00:00
2015-05-24	16:00:00
2015-07-02	12:00:00
2015-09-06	16:00:00
2015-11-25	16:45:00
2015-11-27	12:05:00
2015-12-24	12:05:00
2016-01-17	16:00:00
2016-02-14	16:00:00
2016-05-29	16:00:00
2016-09-04	16:00:00
2016-11-23	16:45:00
2016-11-25	12:05:00
2016-12-23	12:05:00
2017-01-15	16:00:00
2017-02-19	16:00:00
2017-05-28	16:00:00
2017-07-03	12:05:00

2017-09-03	16:00:00
2017-11-22	16:45:00
2017-12-22	12:05:00
2018-01-14	16:00:00
2018-02-18	16:00:00
2018-05-27	16:00:00
2018-07-03	12:05:00
2018-09-02	16:00:00
2018-11-21	16:45:00
2018-11-23	12:05:00
2018-12-24	12:05:00
2019-01-20	16:00:00
2019-02-17	16:00:00
2019-05-26	16:00:00
2019-07-03	12:05:00
2019-09-01	16:00:00
2019-11-27	16:45:00
2019-11-29	12:05:00
2019-12-24	12:05:00
2020-01-19	16:00:00
2020-02-16	16:00:00
2020-05-24	16:00:00
2020-07-02	12:05:00
2020-09-06	16:00:00
2020-11-25	16:45:00
2020-11-27	12:05:00
2020-12-24	12:05:00
2021-01-17	16:00:00

2021-02-14	16:00:00
2021-05-30	16:00:00
2021-09-05	16:00:00
2021-11-24	16:45:00
2021-11-26	12:05:00
2022-01-16	16:00:00
2022-02-20	16:00:00
2022-05-29	16:00:00
2022-06-19	16:00:00
2022-09-04	16:00:00
2022-11-23	16:45:00
2022-11-25	12:05:00
2023-01-15	16:00:00
2023-02-19	16:00:00
2023-05-28	16:00:00
2023-06-18	16:00:00
2023-09-03	16:00:00
2023-11-22	16:00:00
2023-11-24	12:05:00
2024-03-28	16:00:00
2024-12-24	12:05:00
2024-12-31	16:00:00
2025-01-19	16:00:00
2025-02-16	16:00:00
2025-04-17	16:00:00
2025-05-25	16:00:00
2025-06-18	16:45:00
2025-07-03	16:00:00

2025-08-31	16:00:00
2025-11-26	16:00:00
2025-11-28	12:05:00
2025-12-24	12:05:00
2025-12-31	16:00:00
2026-01-18	16:00:00
2026-02-15	16:00:00
2026-04-02	16:00:00
2026-05-24	16:00:00
2026-06-18	16:00:00
2026-07-02	16:00:00
2026-09-06	16:00:00
2026-11-25	16:00:00
2026-11-27	12:05:00
2026-12-24	12:05:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Black Sea Wheat Financially Settled (Platts) Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2011-12-27	09:30:00
2012-01-03	09:30:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-11-23	09:30:00
2013-01-02	09:30:00
2013-01-21	19:00:00
2013-02-18	19:00:00

2013-05-27	19:00:00
2013-07-05	08:30:00
2013-09-02	19:00:00
2013-11-29	08:30:00
2013-12-26	08:30:00
2014-01-02	08:30:00
2014-01-20	19:00:00
2014-02-17	19:00:00
2014-05-26	19:00:00
2014-09-01	19:00:00
2015-01-19	19:00:00
2015-02-16	19:00:00
2015-05-25	19:00:00
2015-09-07	19:00:00
2016-01-18	19:00:00
2016-02-15	19:00:00
2016-05-30	19:00:00
2016-09-05	19:00:00
2016-11-25	08:30:00
2017-01-16	19:00:00
2017-02-20	19:00:00
2017-05-29	19:00:00
2017-07-05	08:30:00
2017-09-04	19:00:00
2017-11-24	08:30:00
2018-01-15	19:00:00
2018-02-19	19:00:00
2018-05-28	19:00:00

2018-07-05	08:30:00
2018-09-03	19:00:00
2018-11-23	08:30:00
2019-01-21	19:00:00
2019-02-18	19:00:00
2019-05-27	19:00:00
2019-07-05	08:30:00
2019-09-02	19:00:00
2019-11-29	08:30:00
2020-01-20	19:00:00
2020-02-17	19:00:00
2020-05-25	19:00:00
2020-09-07	19:00:00
2020-11-27	08:30:00
2021-01-18	19:00:00
2021-02-15	19:00:00
2021-05-31	19:00:00
2021-09-06	19:00:00
2021-11-26	08:30:00
2022-01-17	19:00:00
2022-02-21	19:00:00
2022-05-30	19:00:00
2022-06-20	19:00:00
2022-09-05	19:00:00
2023-01-16	19:00:00
2023-02-20	19:00:00
2023-05-29	19:00:00
2023-06-19	19:00:00

2023-09-04	19:00:00
2024-01-15	19:00:00
2024-02-19	19:00:00
2024-05-27	19:00:00
2024-06-19	19:00:00
2024-09-02	19:00:00
2025-01-20	19:00:00
2025-02-17	19:00:00
2025-05-26	19:00:00
2025-06-19	19:00:00
2025-09-01	19:00:00
2026-01-02	08:30:00
2026-01-19	19:00:00
2026-02-16	19:00:00
2026-05-25	19:00:00
2026-09-07	19:00:00
2026-11-27	08:30:00

Time Zone

The Black Sea Wheat Financially Settled (Platts) Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

EH

Introduction

This page shows the trading hours, holidays, and time zone of the Ethanol Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Ethanol Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Ethanol Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Ethanol Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Ethanol Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)
2023-04-07

Early Closes

The following table shows the early closes for the Ethanol Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-19	16:15:00
2009-02-16	16:15:00
2009-05-22	15:15:00
2009-05-25	12:15:00
2009-07-03	12:30:00
2009-09-04	15:15:00
2009-09-07	12:15:00
2009-10-09	15:15:00
2009-11-26	12:15:00
2009-11-27	12:45:00
2009-12-24	12:45:00
2010-01-15	15:15:00
2010-01-18	12:15:00
2010-02-12	15:15:00
2010-02-15	12:15:00
2010-05-28	15:15:00

2010-05-31	12:15:00
2010-07-02	15:15:00
2010-07-05	12:15:00
2010-09-03	15:15:00
2010-09-06	12:15:00
2010-10-08	15:15:00
2010-11-25	12:15:00
2010-11-26	12:45:00
2010-12-31	15:15:00
2011-01-14	15:15:00
2011-01-17	12:15:00
2011-02-21	12:15:00
2011-05-30	12:15:00
2011-07-04	12:15:00
2011-09-05	12:15:00
2011-11-24	12:15:00
2011-11-25	12:45:00
2012-01-16	12:15:00
2012-02-20	12:15:00
2012-05-28	12:15:00
2012-07-04	12:15:00
2012-09-03	12:15:00
2012-11-22	12:15:00
2012-11-23	12:45:00
2012-12-24	12:45:00
2013-01-21	12:15:00
2013-02-18	12:15:00
2013-05-27	12:15:00

2013-07-04	12:15:00
2013-09-02	12:15:00
2013-11-28	12:15:00
2013-11-29	12:45:00
2013-12-24	12:45:00
2014-01-20	12:15:00
2014-02-17	12:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:45:00
2014-12-24	12:45:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:45:00
2015-12-24	12:45:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:45:00

2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:45:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:45:00
2018-12-24	12:45:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:45:00
2019-12-24	12:45:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00

2020-11-26	12:00:00
2020-11-27	12:45:00
2020-12-24	12:45:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:45:00
2022-01-17	13:30:00
2022-02-21	13:30:00
2022-05-30	13:30:00
2022-06-20	13:30:00
2022-07-04	13:30:00
2022-09-05	13:30:00
2022-11-24	13:30:00
2022-11-25	12:45:00
2023-01-16	13:30:00
2023-02-20	13:30:00
2023-05-29	13:30:00
2023-06-19	13:30:00
2023-07-04	13:30:00
2023-09-04	13:30:00
2023-11-23	13:30:00
2023-11-24	12:45:00
2024-01-15	13:30:00
2024-02-19	13:30:00

2024-03-28	16:00:00
2024-05-27	13:30:00
2024-06-19	13:30:00
2024-07-04	13:30:00
2024-09-02	13:30:00
2024-11-28	13:30:00
2024-12-24	12:45:00
2024-12-31	16:00:00
2025-01-20	13:30:00
2025-02-17	13:30:00
2025-04-17	16:00:00
2025-05-26	13:30:00
2025-06-19	13:30:00
2025-07-04	12:00:00
2025-09-01	13:30:00
2025-11-27	13:30:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-25	16:00:00

Late Opens

The following table shows the late opens for the Ethanol Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00

2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00

2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00

2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-07-03	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00

2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00

Time Zone

The Ethanol Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

F1U

Introduction

This page shows the trading hours, holidays, and time zone of the 5-Year USD MAC Swap Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the 5-Year USD MAC Swap Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the 5-Year USD MAC Swap Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the 5-Year USD MAC Swap Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CBOT Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04

2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-05-29	2023-06-19	2023-07-04

2023-11-23	2023-12-25	2024-01-01	2024-01-15	2024-02-19
2024-03-29	2024-05-27	2024-06-19	2024-07-04	2024-09-02
2024-11-28	2024-12-25	2025-01-09	2025-01-20	2025-04-18

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The 5-Year USD MAC Swap Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

KE

Introduction

This page shows the trading hours, holidays, and time zone of the KC HRW Wheat Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the KC HRW Wheat Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 07:45:00
Tuesday	00:00:00 to 07:45:00
Wednesday	00:00:00 to 07:45:00
Thursday	00:00:00 to 07:45:00
Friday	00:00:00 to 07:45:00

Regular Trading Hours

The following table shows the regular trading hours for the KC HRW Wheat Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 13:20:00
Tuesday	08:30:00 to 13:20:00
Wednesday	08:30:00 to 13:20:00
Thursday	08:30:00 to 13:20:00
Friday	08:30:00 to 13:20:00

Post-market Hours

The following table shows the post-market hours for the KC HRW Wheat Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	19:00:00 to 24:00:00
Tuesday	19:00:00 to 24:00:00
Wednesday	19:00:00 to 24:00:00
Thursday	19:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the KC HRW Wheat Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)				
2013-01-01	2013-01-20	2013-02-17	2013-03-29	2013-05-26
2013-07-04	2013-09-01	2014-01-01	2014-01-19	2014-02-16
2014-04-18	2014-05-25	2014-07-04	2014-07-06	2014-08-31
2015-01-01	2015-01-18	2015-02-15	2015-04-03	2015-05-24
2015-07-03	2015-09-06	2016-01-01	2016-01-17	2016-02-14
2016-03-25	2016-05-29	2016-07-03	2016-07-04	2016-09-04
2017-01-15	2017-02-19	2017-04-14	2017-05-28	2017-07-04
2017-09-03	2018-01-01	2018-01-14	2018-02-18	2018-03-30
2018-05-27	2018-07-04	2018-09-02	2019-01-01	2019-01-20
2019-02-17	2019-04-19	2019-05-26	2019-07-04	2019-09-01
2020-01-01	2020-01-19	2020-02-16	2020-04-10	2020-05-24
2020-07-03	2020-09-06	2021-01-01	2022-12-25	2023-01-01
2023-04-07	2024-01-02	2025-01-01	2025-07-04	2025-11-27
2025-12-25	2026-01-01	2026-04-03	2026-06-19	2026-07-03
2026-11-26	2026-12-25	2027-01-01		

Early Closes

The following table shows the early closes for the KC HRW Wheat Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-25	12:00:00
2012-01-15	19:00:00

2012-02-19	19:00:00
2012-11-21	16:45:00
2012-11-23	12:00:00
2012-12-24	12:00:00
2013-01-20	16:00:00
2013-02-17	16:00:00
2013-05-26	16:00:00
2013-07-03	12:00:00
2013-09-01	16:00:00
2013-11-27	16:45:00
2013-11-29	12:00:00
2013-12-24	12:00:00
2014-01-19	16:00:00
2014-02-16	16:00:00
2014-05-25	16:00:00
2014-07-03	12:00:00
2014-08-31	16:00:00
2014-11-26	16:45:00
2014-11-28	12:00:00
2014-12-24	12:00:00
2015-01-18	16:00:00
2015-02-15	16:00:00
2015-05-24	16:00:00
2015-07-02	12:00:00
2015-09-06	16:00:00
2015-11-25	16:45:00
2015-11-27	12:05:00
2015-12-24	12:05:00

2016-01-17	16:00:00
2016-02-14	16:00:00
2016-05-29	16:00:00
2016-09-04	16:00:00
2016-11-23	16:45:00
2016-11-25	12:05:00
2016-12-23	12:05:00
2017-01-15	16:00:00
2017-02-19	16:00:00
2017-05-28	16:00:00
2017-07-03	12:05:00
2017-09-03	16:00:00
2017-11-22	16:45:00
2017-12-22	12:05:00
2018-01-14	16:00:00
2018-02-18	16:00:00
2018-05-27	16:00:00
2018-07-03	12:05:00
2018-09-02	16:00:00
2018-11-21	16:45:00
2018-11-23	12:05:00
2018-12-24	12:05:00
2019-01-20	16:00:00
2019-02-17	16:00:00
2019-05-26	16:00:00
2019-07-03	12:05:00
2019-09-01	16:00:00
2019-11-27	16:45:00

2019-11-29	12:05:00
2019-12-24	12:05:00
2020-01-19	16:00:00
2020-02-16	16:00:00
2020-05-24	16:00:00
2020-07-02	12:05:00
2020-09-06	16:00:00
2020-11-25	16:45:00
2020-11-27	12:05:00
2020-12-24	12:05:00
2021-01-17	16:00:00
2021-02-14	16:00:00
2021-05-30	16:00:00
2021-09-05	16:00:00
2021-11-24	16:45:00
2021-11-26	12:05:00
2022-01-16	16:00:00
2022-02-20	16:00:00
2022-05-29	16:00:00
2022-06-19	16:00:00
2022-09-04	16:00:00
2022-11-23	16:45:00
2022-11-25	12:05:00
2023-01-15	16:00:00
2023-02-19	16:00:00
2023-05-28	16:00:00
2023-06-18	16:00:00
2023-09-03	16:00:00

2023-11-22	16:00:00
2023-11-24	12:05:00
2024-03-28	16:00:00
2024-12-24	12:05:00
2024-12-31	16:00:00
2025-01-19	16:00:00
2025-02-16	16:00:00
2025-04-17	16:00:00
2025-05-25	16:00:00
2025-06-18	16:45:00
2025-07-03	16:00:00
2025-08-31	16:00:00
2025-11-26	16:00:00
2025-11-28	12:05:00
2025-12-24	12:05:00
2025-12-31	16:00:00
2026-01-18	16:00:00
2026-02-15	16:00:00
2026-04-02	16:00:00
2026-05-24	16:00:00
2026-06-18	16:00:00
2026-07-02	16:00:00
2026-09-06	16:00:00
2026-11-25	16:00:00
2026-11-27	12:05:00
2026-12-24	12:05:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the KC HRW Wheat Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2011-12-27	09:30:00
2012-01-03	09:30:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-11-23	09:30:00
2013-01-02	09:30:00
2013-01-21	19:00:00
2013-02-18	19:00:00
2013-05-27	19:00:00
2013-07-05	08:30:00
2013-09-02	19:00:00
2013-11-29	08:30:00
2013-12-26	08:30:00
2014-01-02	08:30:00
2014-01-20	19:00:00
2014-02-17	19:00:00
2014-05-26	19:00:00
2014-09-01	19:00:00
2015-01-19	19:00:00
2015-02-16	19:00:00
2015-05-25	19:00:00
2015-09-07	19:00:00
2016-01-18	19:00:00
2016-02-15	19:00:00
2016-05-30	19:00:00
2016-09-05	19:00:00

2016-11-25	08:30:00
2017-01-16	19:00:00
2017-02-20	19:00:00
2017-05-29	19:00:00
2017-07-05	08:30:00
2017-09-04	19:00:00
2017-11-24	08:30:00
2018-01-15	19:00:00
2018-02-19	19:00:00
2018-05-28	19:00:00
2018-07-05	08:30:00
2018-09-03	19:00:00
2018-11-23	08:30:00
2019-01-21	19:00:00
2019-02-18	19:00:00
2019-05-27	19:00:00
2019-07-05	08:30:00
2019-09-02	19:00:00
2019-11-29	08:30:00
2020-01-20	19:00:00
2020-02-17	19:00:00
2020-05-25	19:00:00
2020-09-07	19:00:00
2020-11-27	08:30:00
2021-01-18	19:00:00
2021-02-15	19:00:00
2021-05-31	19:00:00
2021-09-06	19:00:00

2021-11-26	08:30:00
2022-01-17	19:00:00
2022-02-21	19:00:00
2022-05-30	19:00:00
2022-06-20	19:00:00
2022-09-05	19:00:00
2023-01-16	19:00:00
2023-02-20	19:00:00
2023-05-29	19:00:00
2023-06-19	19:00:00
2023-09-04	19:00:00
2024-01-15	19:00:00
2024-02-19	19:00:00
2024-05-27	19:00:00
2024-06-19	19:00:00
2024-09-02	19:00:00
2025-01-20	19:00:00
2025-02-17	19:00:00
2025-05-26	19:00:00
2025-06-19	19:00:00
2025-09-01	19:00:00
2026-01-02	08:30:00
2026-01-19	19:00:00
2026-02-16	19:00:00
2026-05-25	19:00:00
2026-09-07	19:00:00
2026-11-27	08:30:00

Time Zone

The KC HRW Wheat Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

MYM

Introduction

This page shows the trading hours, holidays, and time zone of the Micro E-mini Dow Jones Industrial Average Index Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro E-mini Dow Jones Industrial Average Index Futures contract in the CBOT Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro E-mini Dow Jones Industrial Average Index Futures contract in the CBOT Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro E-mini Dow Jones Industrial Average Index Futures contract in the CBOT Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro E-mini Dow Jones Industrial Average Index Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro E-mini Dow Jones Industrial Average Index Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	11:30:00
2009-02-16	11:30:00
2009-05-25	11:30:00
2009-07-03	11:30:00
2009-09-07	11:30:00
2009-11-26	11:30:00
2009-11-27	13:15:00
2009-12-24	13:15:00
2010-01-18	11:30:00
2010-02-15	11:30:00
2010-04-02	09:15:00
2010-05-31	11:30:00
2010-07-05	11:30:00
2010-09-06	11:30:00
2010-11-25	11:30:00

2010-11-26	13:15:00
2011-01-17	11:30:00
2011-02-21	11:30:00
2011-05-30	11:30:00
2011-07-04	11:30:00
2011-09-05	11:30:00
2011-11-24	11:30:00
2011-11-25	13:15:00
2012-01-16	11:30:00
2012-02-20	11:30:00
2012-04-06	09:15:00
2012-05-28	11:30:00
2012-07-03	13:15:00
2012-07-04	11:30:00
2012-09-03	11:30:00
2012-11-22	11:30:00
2012-11-23	13:15:00
2012-12-24	13:15:00
2013-01-21	11:30:00
2013-02-18	11:30:00
2013-05-27	11:30:00
2013-07-03	13:15:00
2013-07-04	11:30:00
2013-09-02	11:30:00
2013-11-28	11:30:00
2013-11-29	13:15:00
2013-12-24	13:15:00
2014-01-20	11:30:00

2014-02-17	11:30:00
2014-05-26	13:00:00
2014-07-03	13:15:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:15:00
2014-12-24	13:15:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-04-03	09:15:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:15:00
2015-12-24	13:15:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:15:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-03	13:15:00

2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:15:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-03	13:15:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:15:00
2018-12-24	13:15:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-03	13:15:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:15:00
2019-12-24	13:15:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00
2020-11-26	13:00:00

2020-11-27	13:15:00
2020-12-24	13:15:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-04-02	09:15:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:15:00
2022-01-17	13:00:00
2022-02-21	13:00:00
2022-05-30	13:00:00
2022-06-20	13:00:00
2022-07-04	13:00:00
2022-09-05	13:00:00
2022-11-24	13:00:00
2022-11-25	13:15:00
2023-01-16	13:00:00
2023-02-20	13:00:00
2023-04-07	09:15:00
2023-05-29	13:00:00
2023-06-19	13:00:00
2023-07-04	13:00:00
2023-09-04	13:00:00
2023-11-23	13:00:00
2023-11-24	13:15:00
2024-01-15	13:00:00

2024-02-19	13:00:00
2024-05-27	13:00:00
2024-06-19	13:00:00
2024-07-04	13:00:00
2024-09-02	13:00:00
2024-11-28	13:00:00
2024-12-24	13:15:00
2024-12-31	17:00:00
2025-01-20	13:00:00
2025-02-17	13:00:00
2025-04-17	17:00:00
2025-05-26	13:00:00
2025-06-19	13:00:00
2025-07-03	13:15:00
2025-07-04	13:00:00
2025-09-01	13:00:00
2025-11-27	13:00:00
2025-11-28	13:15:00
2025-12-24	13:15:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	13:00:00
2026-02-16	13:00:00
2026-04-03	09:15:00
2026-05-25	13:00:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	13:00:00

2026-11-26	13:00:00
2026-11-27	13:15:00
2026-12-24	13:15:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Micro E-mini Dow Jones Industrial Average Index Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-07-03	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Micro E-mini Dow Jones Industrial Average Index Futures contract in the CBOT Future market trades in the `America/New York` time zone.

CBOT

TN

Introduction

This page shows the trading hours, holidays, and time zone of the Ultra 10-Year U.S. Treasury Note Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Ultra 10-Year U.S. Treasury Note Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Ultra 10-Year U.S. Treasury Note Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Ultra 10-Year U.S. Treasury Note Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Ultra 10-Year U.S. Treasury Note Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Ultra 10-Year U.S. Treasury Note Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	10:30:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00

2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00

2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00

2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00
2023-01-16	12:00:00
2023-02-20	12:00:00
2023-04-07	10:15:00
2023-05-29	12:00:00
2023-06-19	12:00:00
2023-07-04	12:00:00
2023-09-04	12:00:00
2023-11-23	12:00:00
2023-11-24	12:15:00

2024-01-15	12:00:00
2024-02-19	12:00:00
2024-03-28	16:00:00
2024-05-27	12:00:00
2024-06-19	12:00:00
2024-07-04	12:00:00
2024-09-02	12:00:00
2024-11-28	12:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	12:00:00
2025-02-17	12:00:00
2025-04-17	16:00:00
2025-05-26	12:00:00
2025-06-19	12:00:00
2025-07-04	12:00:00
2025-09-01	12:00:00
2025-11-27	12:00:00
2025-11-28	12:15:00
2025-12-24	12:15:00
2025-12-25	16:00:00
2025-12-31	16:00:00
2026-01-19	12:00:00
2026-02-16	12:00:00
2026-04-03	10:15:00
2026-05-25	12:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00

2026-09-07	12:00:00
2026-11-26	12:00:00
2026-11-27	12:15:00
2026-12-24	12:15:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Ultra 10-Year U.S. Treasury Note Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00

2012-07-04	17:00:00
2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00

2017-07-04	17:00:00
2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-07-03	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00

2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00

2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Ultra 10-Year U.S. Treasury Note Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

UB

Introduction

This page shows the trading hours, holidays, and time zone of the Ultra U.S. Treasury Bond Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Ultra U.S. Treasury Bond Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Ultra U.S. Treasury Bond Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Ultra U.S. Treasury Bond Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Ultra U.S. Treasury Bond Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Ultra U.S. Treasury Bond Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	10:30:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00
2023-01-16	12:00:00
2023-02-20	12:00:00
2023-04-07	10:15:00
2023-05-29	12:00:00
2023-06-19	12:00:00
2023-07-04	12:00:00
2023-09-04	12:00:00
2023-11-23	12:00:00
2023-11-24	12:15:00
2024-01-15	12:00:00

2024-02-19	12:00:00
2024-03-28	16:00:00
2024-05-27	12:00:00
2024-06-19	12:00:00
2024-07-04	12:00:00
2024-09-02	12:00:00
2024-11-28	12:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	12:00:00
2025-02-17	12:00:00
2025-04-17	16:00:00
2025-05-26	12:00:00
2025-06-19	12:00:00
2025-07-04	12:00:00
2025-09-01	12:00:00
2025-11-27	12:00:00
2025-11-28	12:15:00
2025-12-24	12:15:00
2025-12-25	16:00:00
2025-12-31	16:00:00
2026-01-19	12:00:00
2026-02-16	12:00:00
2026-04-03	10:15:00
2026-05-25	12:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	12:00:00

2026-11-26	12:00:00
2026-11-27	12:15:00
2026-12-24	12:15:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Ultra U.S. Treasury Bond Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00

2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00

2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-07-03	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00

2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00

2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Ultra U.S. Treasury Bond Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

YM

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini Dow ($5) Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini Dow ($5) Futures contract in the CBOT Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini Dow ($5) Futures contract in the CBOT Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini Dow ($5) Futures contract in the CBOT Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the E-mini Dow ($5) Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the E-mini Dow ($5) Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	11:30:00
2009-02-16	11:30:00
2009-05-25	11:30:00
2009-07-03	11:30:00
2009-09-07	11:30:00
2009-11-26	11:30:00
2009-11-27	13:15:00
2009-12-24	13:15:00
2010-01-18	11:30:00
2010-02-15	11:30:00
2010-04-02	09:15:00
2010-05-31	11:30:00
2010-07-05	11:30:00
2010-09-06	11:30:00
2010-11-25	11:30:00
2010-11-26	13:15:00

2011-01-17	11:30:00
2011-02-21	11:30:00
2011-05-30	11:30:00
2011-07-04	11:30:00
2011-09-05	11:30:00
2011-11-24	11:30:00
2011-11-25	13:15:00
2012-01-16	11:30:00
2012-02-20	11:30:00
2012-04-06	09:15:00
2012-05-28	11:30:00
2012-07-03	13:15:00
2012-07-04	11:30:00
2012-09-03	11:30:00
2012-11-22	11:30:00
2012-11-23	13:15:00
2012-12-24	13:15:00
2013-01-21	11:30:00
2013-02-18	11:30:00
2013-05-27	11:30:00
2013-07-03	13:15:00
2013-07-04	11:30:00
2013-09-02	11:30:00
2013-11-28	11:30:00
2013-11-29	13:15:00
2013-12-24	13:15:00
2014-01-20	11:30:00
2014-02-17	11:30:00

2014-05-26	13:00:00
2014-07-03	13:15:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:15:00
2014-12-24	13:15:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-04-03	09:15:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:15:00
2015-12-24	13:15:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:15:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-03	13:15:00
2017-07-04	13:00:00

2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:15:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-03	13:15:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:15:00
2018-12-24	13:15:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-03	13:15:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:15:00
2019-12-24	13:15:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:15:00

2020-12-24	13:15:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-04-02	09:15:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:15:00
2022-01-17	13:00:00
2022-02-21	13:00:00
2022-05-30	13:00:00
2022-06-20	13:00:00
2022-07-04	13:00:00
2022-09-05	13:00:00
2022-11-24	13:00:00
2022-11-25	13:15:00
2023-01-16	13:00:00
2023-02-20	13:00:00
2023-04-07	09:15:00
2023-05-29	13:00:00
2023-06-19	13:00:00
2023-07-04	13:00:00
2023-09-04	13:00:00
2023-11-23	13:00:00
2023-11-24	13:15:00
2024-01-15	13:00:00
2024-02-19	13:00:00

2024-05-27	13:00:00
2024-06-19	13:00:00
2024-07-04	13:00:00
2024-09-02	13:00:00
2024-11-28	13:00:00
2024-12-24	13:15:00
2024-12-31	17:00:00
2025-01-20	13:00:00
2025-02-17	13:00:00
2025-04-17	17:00:00
2025-05-26	13:00:00
2025-06-19	13:00:00
2025-07-03	13:15:00
2025-07-04	13:00:00
2025-09-01	13:00:00
2025-11-27	13:00:00
2025-11-28	13:15:00
2025-12-24	13:15:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	13:00:00
2026-02-16	13:00:00
2026-04-03	09:15:00
2026-05-25	13:00:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	13:00:00
2026-11-26	13:00:00

2026-11-27	13:15:00
2026-12-24	13:15:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the E-mini Dow ($5) Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00

2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-07-03	18:00:00
2025-09-01	18:00:00

2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The E-mini Dow ($5) Futures contract in the CBOT Future market trades in the `America/New York` time zone.

CBOT

ZB

Introduction

This page shows the trading hours, holidays, and time zone of the U.S. Treasury Bond Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the U.S. Treasury Bond Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the U.S. Treasury Bond Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the U.S. Treasury Bond Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the U.S. Treasury Bond Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the U.S. Treasury Bond Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	10:30:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00
2023-01-16	12:00:00
2023-02-20	12:00:00
2023-04-07	10:15:00
2023-05-29	12:00:00
2023-06-19	12:00:00
2023-07-04	12:00:00
2023-09-04	12:00:00
2023-11-23	12:00:00
2023-11-24	12:15:00
2024-01-15	12:00:00

2024-02-19	12:00:00
2024-03-28	16:00:00
2024-05-27	12:00:00
2024-06-19	12:00:00
2024-07-04	12:00:00
2024-09-02	12:00:00
2024-11-28	12:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	12:00:00
2025-02-17	12:00:00
2025-04-17	16:00:00
2025-05-26	12:00:00
2025-06-19	12:00:00
2025-07-04	12:00:00
2025-09-01	12:00:00
2025-11-27	12:00:00
2025-11-28	12:15:00
2025-12-24	12:15:00
2025-12-25	16:00:00
2025-12-31	16:00:00
2026-01-19	12:00:00
2026-02-16	12:00:00
2026-04-03	10:15:00
2026-05-25	12:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	12:00:00

2026-11-26	12:00:00
2026-11-27	12:15:00
2026-12-24	12:15:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the U.S. Treasury Bond Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00

2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00

2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-07-03	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00

2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00

2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The U.S. Treasury Bond Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

ZC

Introduction

This page shows the trading hours, holidays, and time zone of the Corn Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Corn Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 07:45:00
Tuesday	00:00:00 to 07:45:00
Wednesday	00:00:00 to 07:45:00
Thursday	00:00:00 to 07:45:00
Friday	00:00:00 to 07:45:00

Regular Trading Hours

The following table shows the regular trading hours for the Corn Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 13:20:00
Tuesday	08:30:00 to 13:20:00
Wednesday	08:30:00 to 13:20:00
Thursday	08:30:00 to 13:20:00
Friday	08:30:00 to 13:20:00

Post-market Hours

The following table shows the post-market hours for the Corn Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	19:00:00 to 24:00:00
Tuesday	19:00:00 to 24:00:00
Wednesday	19:00:00 to 24:00:00
Thursday	19:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Corn Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)				
2023-04-07	2024-01-02	2025-01-01	2025-07-04	2025-11-27
2025-12-25	2026-01-01	2026-04-03	2026-06-19	2026-07-03
2026-11-26	2026-12-25	2027-01-01		

Early Closes

The following table shows the early closes for the Corn Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-25	12:00:00
2012-01-15	19:00:00
2012-02-19	19:00:00
2012-11-21	16:45:00
2012-11-23	12:00:00
2012-12-24	12:00:00
2013-01-20	16:00:00
2013-02-17	16:00:00
2013-05-26	16:00:00
2013-07-03	12:00:00
2013-09-01	16:00:00
2013-11-27	16:45:00
2013-11-29	12:00:00
2013-12-24	12:00:00

2014-01-19	16:00:00
2014-02-16	16:00:00
2014-05-25	16:00:00
2014-07-03	12:00:00
2014-08-31	16:00:00
2014-11-26	16:45:00
2014-11-28	12:00:00
2014-12-24	12:00:00
2015-01-18	16:00:00
2015-02-15	16:00:00
2015-05-24	16:00:00
2015-07-02	12:00:00
2015-09-06	16:00:00
2015-11-25	16:45:00
2015-11-27	12:05:00
2015-12-24	12:05:00
2016-01-17	16:00:00
2016-02-14	16:00:00
2016-05-29	16:00:00
2016-09-04	16:00:00
2016-11-23	16:45:00
2016-11-25	12:05:00
2016-12-23	12:05:00
2017-01-15	16:00:00
2017-02-19	16:00:00
2017-05-28	16:00:00
2017-07-03	12:05:00
2017-09-03	16:00:00

2017-11-22	16:45:00
2017-12-22	12:05:00
2018-01-14	16:00:00
2018-02-18	16:00:00
2018-05-27	16:00:00
2018-07-03	12:05:00
2018-09-02	16:00:00
2018-11-21	16:45:00
2018-11-23	12:05:00
2018-12-24	12:05:00
2019-01-20	16:00:00
2019-02-17	16:00:00
2019-05-26	16:00:00
2019-07-03	12:05:00
2019-09-01	16:00:00
2019-11-27	16:45:00
2019-11-29	12:05:00
2019-12-24	12:05:00
2020-01-19	16:00:00
2020-02-16	16:00:00
2020-05-24	16:00:00
2020-07-02	12:05:00
2020-09-06	16:00:00
2020-11-25	16:45:00
2020-11-27	12:05:00
2020-12-24	12:05:00
2021-01-17	16:00:00
2021-02-14	16:00:00

2021-05-30	16:00:00
2021-09-05	16:00:00
2021-11-24	16:45:00
2021-11-26	12:05:00
2022-01-16	16:00:00
2022-02-20	16:00:00
2022-05-29	16:00:00
2022-06-19	16:00:00
2022-09-04	16:00:00
2022-11-23	16:45:00
2022-11-25	12:05:00
2023-01-15	16:00:00
2023-02-19	16:00:00
2023-05-28	16:00:00
2023-06-18	16:00:00
2023-09-03	16:00:00
2023-11-22	16:00:00
2023-11-24	12:05:00
2024-03-28	16:00:00
2024-12-24	12:05:00
2024-12-31	16:00:00
2025-01-19	16:00:00
2025-02-16	16:00:00
2025-04-17	16:00:00
2025-05-25	16:00:00
2025-06-18	16:45:00
2025-07-03	16:00:00
2025-08-31	16:00:00

2025-11-26	16:00:00
2025-11-28	12:05:00
2025-12-24	12:05:00
2025-12-31	16:00:00
2026-01-18	16:00:00
2026-02-15	16:00:00
2026-04-02	16:00:00
2026-05-24	16:00:00
2026-06-18	16:00:00
2026-07-02	16:00:00
2026-09-06	16:00:00
2026-11-25	16:00:00
2026-11-27	12:05:00
2026-12-24	12:05:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Corn Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2011-12-27	09:30:00
2012-01-03	09:30:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-11-23	09:30:00
2013-01-02	09:30:00
2013-01-21	19:00:00
2013-02-18	19:00:00
2013-05-27	19:00:00
2013-07-05	08:30:00

2013-09-02	19:00:00
2013-11-29	08:30:00
2013-12-26	08:30:00
2014-01-02	08:30:00
2014-01-20	19:00:00
2014-02-17	19:00:00
2014-05-26	19:00:00
2014-09-01	19:00:00
2015-01-19	19:00:00
2015-02-16	19:00:00
2015-05-25	19:00:00
2015-09-07	19:00:00
2016-01-18	19:00:00
2016-02-15	19:00:00
2016-05-30	19:00:00
2016-09-05	19:00:00
2016-11-25	08:30:00
2017-01-16	19:00:00
2017-02-20	19:00:00
2017-05-29	19:00:00
2017-07-05	08:30:00
2017-09-04	19:00:00
2017-11-24	08:30:00
2018-01-15	19:00:00
2018-02-19	19:00:00
2018-05-28	19:00:00
2018-07-05	08:30:00
2018-09-03	19:00:00

2018-11-23	08:30:00
2019-01-21	19:00:00
2019-02-18	19:00:00
2019-05-27	19:00:00
2019-07-05	08:30:00
2019-09-02	19:00:00
2019-11-29	08:30:00
2020-01-20	19:00:00
2020-02-17	19:00:00
2020-05-25	19:00:00
2020-09-07	19:00:00
2020-11-27	08:30:00
2021-01-18	19:00:00
2021-02-15	19:00:00
2021-05-31	19:00:00
2021-09-06	19:00:00
2021-11-26	08:30:00
2022-01-17	19:00:00
2022-02-21	19:00:00
2022-05-30	19:00:00
2022-06-20	19:00:00
2022-09-05	19:00:00
2023-01-16	19:00:00
2023-02-20	19:00:00
2023-05-29	19:00:00
2023-06-19	19:00:00
2023-09-04	19:00:00
2024-01-15	19:00:00

2024-02-19	19:00:00
2024-05-27	19:00:00
2024-06-19	19:00:00
2024-09-02	19:00:00
2025-01-20	19:00:00
2025-02-17	19:00:00
2025-05-26	19:00:00
2025-06-19	19:00:00
2025-09-01	19:00:00
2026-01-02	08:30:00
2026-01-19	19:00:00
2026-02-16	19:00:00
2026-05-25	19:00:00
2026-09-07	19:00:00
2026-11-27	08:30:00

Time Zone

The Corn Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

ZF

Introduction

This page shows the trading hours, holidays, and time zone of the 5-Year T-Note Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the 5-Year T-Note Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the 5-Year T-Note Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the 5-Year T-Note Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the 5-Year T-Note Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the 5-Year T-Note Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	10:30:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00
2023-01-16	12:00:00
2023-02-20	12:00:00
2023-04-07	10:15:00
2023-05-29	12:00:00
2023-06-19	12:00:00
2023-07-04	12:00:00
2023-09-04	12:00:00
2023-11-23	12:00:00
2023-11-24	12:15:00
2024-01-15	12:00:00

2024-02-19	12:00:00
2024-03-28	16:00:00
2024-05-27	12:00:00
2024-06-19	12:00:00
2024-07-04	12:00:00
2024-09-02	12:00:00
2024-11-28	12:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	12:00:00
2025-02-17	12:00:00
2025-04-17	16:00:00
2025-05-26	12:00:00
2025-06-19	12:00:00
2025-07-04	12:00:00
2025-09-01	12:00:00
2025-11-27	12:00:00
2025-11-28	12:15:00
2025-12-24	12:15:00
2025-12-25	16:00:00
2025-12-31	16:00:00
2026-01-19	12:00:00
2026-02-16	12:00:00
2026-04-03	10:15:00
2026-05-25	12:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	12:00:00

2026-11-26	12:00:00
2026-11-27	12:15:00
2026-12-24	12:15:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the 5-Year T-Note Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00

2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00

2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-07-03	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00

2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00

2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The 5-Year T-Note Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

ZL

Introduction

This page shows the trading hours, holidays, and time zone of the Soybean Oil Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Soybean Oil Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 07:45:00
Tuesday	00:00:00 to 07:45:00
Wednesday	00:00:00 to 07:45:00
Thursday	00:00:00 to 07:45:00
Friday	00:00:00 to 07:45:00

Regular Trading Hours

The following table shows the regular trading hours for the Soybean Oil Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 13:20:00
Tuesday	08:30:00 to 13:20:00
Wednesday	08:30:00 to 13:20:00
Thursday	08:30:00 to 13:20:00
Friday	08:30:00 to 13:20:00

Post-market Hours

The following table shows the post-market hours for the Soybean Oil Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	19:00:00 to 24:00:00
Tuesday	19:00:00 to 24:00:00
Wednesday	19:00:00 to 24:00:00
Thursday	19:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Soybean Oil Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)				
2023-04-07	2026-01-01	2026-04-03	2026-06-19	2026-07-03
2026-11-26	2026-12-25	2027-01-01		

Early Closes

The following table shows the early closes for the Soybean Oil Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2025-12-31	16:00:00
2026-01-18	16:00:00
2026-02-15	16:00:00
2026-04-02	16:00:00
2026-05-24	16:00:00
2026-06-18	16:00:00
2026-07-02	16:00:00
2026-09-06	16:00:00
2026-11-25	16:00:00
2026-11-27	12:05:00
2026-12-24	12:05:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Soybean Oil Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/Chicago)
2026-01-02	08:30:00
2026-01-19	19:00:00
2026-02-16	19:00:00
2026-05-25	19:00:00
2026-09-07	19:00:00
2026-11-27	08:30:00

Time Zone

The Soybean Oil Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

ZM

Introduction

This page shows the trading hours, holidays, and time zone of the Soybean Meal Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Soybean Meal Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 07:45:00
Tuesday	00:00:00 to 07:45:00
Wednesday	00:00:00 to 07:45:00
Thursday	00:00:00 to 07:45:00
Friday	00:00:00 to 07:45:00

Regular Trading Hours

The following table shows the regular trading hours for the Soybean Meal Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 13:20:00
Tuesday	08:30:00 to 13:20:00
Wednesday	08:30:00 to 13:20:00
Thursday	08:30:00 to 13:20:00
Friday	08:30:00 to 13:20:00

Post-market Hours

The following table shows the post-market hours for the Soybean Meal Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	19:00:00 to 24:00:00
Tuesday	19:00:00 to 24:00:00
Wednesday	19:00:00 to 24:00:00
Thursday	19:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Soybean Meal Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)				
2023-04-07	2026-01-01	2026-04-03	2026-06-19	2026-07-03
2026-11-26	2026-12-25	2027-01-01		

Early Closes

The following table shows the early closes for the Soybean Meal Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2025-12-31	16:00:00
2026-01-18	16:00:00
2026-02-15	16:00:00
2026-04-02	16:00:00
2026-05-24	16:00:00
2026-06-18	16:00:00
2026-07-02	16:00:00
2026-09-06	16:00:00
2026-11-25	16:00:00
2026-11-27	12:05:00
2026-12-24	12:05:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Soybean Meal Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/Chicago)
2026-01-02	08:30:00
2026-01-19	19:00:00
2026-02-16	19:00:00
2026-05-25	19:00:00
2026-09-07	19:00:00
2026-11-27	08:30:00

Time Zone

The Soybean Meal Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

ZN

Introduction

This page shows the trading hours, holidays, and time zone of the 10-Year T-Note Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the 10-Year T-Note Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the 10-Year T-Note Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the 10-Year T-Note Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the 10-Year T-Note Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the 10-Year T-Note Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	10:30:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00
2023-01-16	12:00:00
2023-02-20	12:00:00
2023-04-07	10:15:00
2023-05-29	12:00:00
2023-06-19	12:00:00
2023-07-04	12:00:00
2023-09-04	12:00:00
2023-11-23	12:00:00
2023-11-24	12:15:00
2024-01-15	12:00:00

2024-02-19	12:00:00
2024-03-28	16:00:00
2024-05-27	12:00:00
2024-06-19	12:00:00
2024-07-04	12:00:00
2024-09-02	12:00:00
2024-11-28	12:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	12:00:00
2025-02-17	12:00:00
2025-04-17	16:00:00
2025-05-26	12:00:00
2025-06-19	12:00:00
2025-07-04	12:00:00
2025-09-01	12:00:00
2025-11-27	12:00:00
2025-11-28	12:15:00
2025-12-24	12:15:00
2025-12-25	16:00:00
2025-12-31	16:00:00
2026-01-19	12:00:00
2026-02-16	12:00:00
2026-04-03	10:15:00
2026-05-25	12:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	12:00:00

2026-11-26	12:00:00
2026-11-27	12:15:00
2026-12-24	12:15:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the 10-Year T-Note Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00

2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00

2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-07-03	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00

2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00

2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The 10-Year T-Note Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

ZO

Introduction

This page shows the trading hours, holidays, and time zone of the Oats Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Oats Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 07:45:00
Tuesday	00:00:00 to 07:45:00
Wednesday	00:00:00 to 07:45:00
Thursday	00:00:00 to 07:45:00
Friday	00:00:00 to 07:45:00

Regular Trading Hours

The following table shows the regular trading hours for the Oats Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 13:20:00
Tuesday	08:30:00 to 13:20:00
Wednesday	08:30:00 to 13:20:00
Thursday	08:30:00 to 13:20:00
Friday	08:30:00 to 13:20:00

Post-market Hours

The following table shows the post-market hours for the Oats Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	19:00:00 to 24:00:00
Tuesday	19:00:00 to 24:00:00
Wednesday	19:00:00 to 24:00:00
Thursday	19:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Oats Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)				
2023-04-07	2024-01-02	2025-01-01	2025-07-04	2025-11-27
2025-12-25	2026-01-01	2026-04-03	2026-06-19	2026-07-03
2026-11-26	2026-12-25	2027-01-01		

Early Closes

The following table shows the early closes for the Oats Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-25	12:00:00
2012-01-15	19:00:00
2012-02-19	19:00:00
2012-11-21	16:45:00
2012-11-23	12:00:00
2012-12-24	12:00:00
2013-01-20	16:00:00
2013-02-17	16:00:00
2013-05-26	16:00:00
2013-07-03	12:00:00
2013-09-01	16:00:00
2013-11-27	16:45:00
2013-11-29	12:00:00
2013-12-24	12:00:00

2014-01-19	16:00:00
2014-02-16	16:00:00
2014-05-25	16:00:00
2014-07-03	12:00:00
2014-08-31	16:00:00
2014-11-26	16:45:00
2014-11-28	12:00:00
2014-12-24	12:00:00
2015-01-18	16:00:00
2015-02-15	16:00:00
2015-05-24	16:00:00
2015-07-02	12:00:00
2015-09-06	16:00:00
2015-11-25	16:45:00
2015-11-27	12:05:00
2015-12-24	12:05:00
2016-01-17	16:00:00
2016-02-14	16:00:00
2016-05-29	16:00:00
2016-09-04	16:00:00
2016-11-23	16:45:00
2016-11-25	12:05:00
2016-12-23	12:05:00
2017-01-15	16:00:00
2017-02-19	16:00:00
2017-05-28	16:00:00
2017-07-03	12:05:00
2017-09-03	16:00:00

2017-11-22	16:45:00
2017-12-22	12:05:00
2018-01-14	16:00:00
2018-02-18	16:00:00
2018-05-27	16:00:00
2018-07-03	12:05:00
2018-09-02	16:00:00
2018-11-21	16:45:00
2018-11-23	12:05:00
2018-12-24	12:05:00
2019-01-20	16:00:00
2019-02-17	16:00:00
2019-05-26	16:00:00
2019-07-03	12:05:00
2019-09-01	16:00:00
2019-11-27	16:45:00
2019-11-29	12:05:00
2019-12-24	12:05:00
2020-01-19	16:00:00
2020-02-16	16:00:00
2020-05-24	16:00:00
2020-07-02	12:05:00
2020-09-06	16:00:00
2020-11-25	16:45:00
2020-11-27	12:05:00
2020-12-24	12:05:00
2021-01-17	16:00:00
2021-02-14	16:00:00

2021-05-30	16:00:00
2021-09-05	16:00:00
2021-11-24	16:45:00
2021-11-26	12:05:00
2022-01-16	16:00:00
2022-02-20	16:00:00
2022-05-29	16:00:00
2022-06-19	16:00:00
2022-09-04	16:00:00
2022-11-23	16:45:00
2022-11-25	12:05:00
2023-01-15	16:00:00
2023-02-19	16:00:00
2023-05-28	16:00:00
2023-06-18	16:00:00
2023-09-03	16:00:00
2023-11-22	16:00:00
2023-11-24	12:05:00
2024-03-28	16:00:00
2024-12-24	12:05:00
2024-12-31	16:00:00
2025-01-19	16:00:00
2025-02-16	16:00:00
2025-04-17	16:00:00
2025-05-25	16:00:00
2025-06-18	16:45:00
2025-07-03	16:00:00
2025-08-31	16:00:00

2025-11-26	16:00:00
2025-11-28	12:05:00
2025-12-24	12:05:00
2025-12-31	16:00:00
2026-01-18	16:00:00
2026-02-15	16:00:00
2026-04-02	16:00:00
2026-05-24	16:00:00
2026-06-18	16:00:00
2026-07-02	16:00:00
2026-09-06	16:00:00
2026-11-25	16:00:00
2026-11-27	12:05:00
2026-12-24	12:05:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Oats Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2011-12-27	09:30:00
2012-01-03	09:30:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-11-23	09:30:00
2013-01-02	09:30:00
2013-01-21	19:00:00
2013-02-18	19:00:00
2013-05-27	19:00:00
2013-07-05	08:30:00

2013-09-02	19:00:00
2013-11-29	08:30:00
2013-12-26	08:30:00
2014-01-02	08:30:00
2014-01-20	19:00:00
2014-02-17	19:00:00
2014-05-26	19:00:00
2014-09-01	19:00:00
2015-01-19	19:00:00
2015-02-16	19:00:00
2015-05-25	19:00:00
2015-09-07	19:00:00
2016-01-18	19:00:00
2016-02-15	19:00:00
2016-05-30	19:00:00
2016-09-05	19:00:00
2016-11-25	08:30:00
2017-01-16	19:00:00
2017-02-20	19:00:00
2017-05-29	19:00:00
2017-07-05	08:30:00
2017-09-04	19:00:00
2017-11-24	08:30:00
2018-01-15	19:00:00
2018-02-19	19:00:00
2018-05-28	19:00:00
2018-07-05	08:30:00
2018-09-03	19:00:00

2018-11-23	08:30:00
2019-01-21	19:00:00
2019-02-18	19:00:00
2019-05-27	19:00:00
2019-07-05	08:30:00
2019-09-02	19:00:00
2019-11-29	08:30:00
2020-01-20	19:00:00
2020-02-17	19:00:00
2020-05-25	19:00:00
2020-09-07	19:00:00
2020-11-27	08:30:00
2021-01-18	19:00:00
2021-02-15	19:00:00
2021-05-31	19:00:00
2021-09-06	19:00:00
2021-11-26	08:30:00
2022-01-17	19:00:00
2022-02-21	19:00:00
2022-05-30	19:00:00
2022-06-20	19:00:00
2022-09-05	19:00:00
2023-01-16	19:00:00
2023-02-20	19:00:00
2023-05-29	19:00:00
2023-06-19	19:00:00
2023-09-04	19:00:00
2024-01-15	19:00:00

2024-02-19	19:00:00
2024-05-27	19:00:00
2024-06-19	19:00:00
2024-09-02	19:00:00
2025-01-20	19:00:00
2025-02-17	19:00:00
2025-05-26	19:00:00
2025-06-19	19:00:00
2025-09-01	19:00:00
2026-01-02	08:30:00
2026-01-19	19:00:00
2026-02-16	19:00:00
2026-05-25	19:00:00
2026-09-07	19:00:00
2026-11-27	08:30:00

Time Zone

The Oats Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

ZS

Introduction

This page shows the trading hours, holidays, and time zone of the Soybean Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Soybean Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 07:45:00
Tuesday	00:00:00 to 07:45:00
Wednesday	00:00:00 to 07:45:00
Thursday	00:00:00 to 07:45:00
Friday	00:00:00 to 07:45:00

Regular Trading Hours

The following table shows the regular trading hours for the Soybean Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 13:20:00
Tuesday	08:30:00 to 13:20:00
Wednesday	08:30:00 to 13:20:00
Thursday	08:30:00 to 13:20:00
Friday	08:30:00 to 13:20:00

Post-market Hours

The following table shows the post-market hours for the Soybean Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	19:00:00 to 24:00:00
Tuesday	19:00:00 to 24:00:00
Wednesday	19:00:00 to 24:00:00
Thursday	19:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Soybean Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)				
2023-04-07	2026-01-01	2026-04-03	2026-06-19	2026-07-03
2026-11-26	2026-12-25	2027-01-01		

Early Closes

The following table shows the early closes for the Soybean Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2025-12-31	16:00:00
2026-01-18	16:00:00
2026-02-15	16:00:00
2026-04-02	16:00:00
2026-05-24	16:00:00
2026-06-18	16:00:00
2026-07-02	16:00:00
2026-09-06	16:00:00
2026-11-25	16:00:00
2026-11-27	12:05:00
2026-12-24	12:05:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Soybean Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/Chicago)
2026-01-02	08:30:00
2026-01-19	19:00:00
2026-02-16	19:00:00
2026-05-25	19:00:00
2026-09-07	19:00:00
2026-11-27	08:30:00

Time Zone

The Soybean Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

ZT

Introduction

This page shows the trading hours, holidays, and time zone of the 2-Year T-Note Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the 2-Year T-Note Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the 2-Year T-Note Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the 2-Year T-Note Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the 2-Year T-Note Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the 2-Year T-Note Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	10:30:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00
2023-01-16	12:00:00
2023-02-20	12:00:00
2023-04-07	10:15:00
2023-05-29	12:00:00
2023-06-19	12:00:00
2023-07-04	12:00:00
2023-09-04	12:00:00
2023-11-23	12:00:00
2023-11-24	12:15:00
2024-01-15	12:00:00

2024-02-19	12:00:00
2024-03-28	16:00:00
2024-05-27	12:00:00
2024-06-19	12:00:00
2024-07-04	12:00:00
2024-09-02	12:00:00
2024-11-28	12:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	12:00:00
2025-02-17	12:00:00
2025-04-17	16:00:00
2025-05-26	12:00:00
2025-06-19	12:00:00
2025-07-04	12:00:00
2025-09-01	12:00:00
2025-11-27	12:00:00
2025-11-28	12:15:00
2025-12-24	12:15:00
2025-12-25	16:00:00
2025-12-31	16:00:00
2026-01-19	12:00:00
2026-02-16	12:00:00
2026-04-03	10:15:00
2026-05-25	12:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	12:00:00

2026-11-26	12:00:00
2026-11-27	12:15:00
2026-12-24	12:15:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the 2-Year T-Note Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00

2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00

2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-07-03	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00

2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00

2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The 2-Year T-Note Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

CBOT

ZW

Introduction

This page shows the trading hours, holidays, and time zone of the Chicago SRW Wheat Futures contract in the CBOT Future market.

Pre-market Hours

The following table shows the pre-market hours for the Chicago SRW Wheat Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 07:45:00
Tuesday	00:00:00 to 07:45:00
Wednesday	00:00:00 to 07:45:00
Thursday	00:00:00 to 07:45:00
Friday	00:00:00 to 07:45:00

Regular Trading Hours

The following table shows the regular trading hours for the Chicago SRW Wheat Futures contract in the CBOT Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 13:20:00
Tuesday	08:30:00 to 13:20:00
Wednesday	08:30:00 to 13:20:00
Thursday	08:30:00 to 13:20:00
Friday	08:30:00 to 13:20:00

Post-market Hours

The following table shows the post-market hours for the Chicago SRW Wheat Futures contract in the CBOT Future market:

Weekday	Time (America/Chicago)
Monday	19:00:00 to 24:00:00
Tuesday	19:00:00 to 24:00:00
Wednesday	19:00:00 to 24:00:00
Thursday	19:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Chicago SRW Wheat Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)				
2023-04-07	2024-01-02	2025-01-01	2025-07-04	2025-11-27
2025-12-25	2026-01-01	2026-04-03	2026-06-19	2026-07-03
2026-11-26	2026-12-25	2027-01-01		

Early Closes

The following table shows the early closes for the Chicago SRW Wheat Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2011-11-25	12:00:00
2012-01-15	19:00:00
2012-02-19	19:00:00
2012-11-21	16:45:00
2012-11-23	12:00:00
2012-12-24	12:00:00
2013-01-20	16:00:00
2013-02-17	16:00:00
2013-05-26	16:00:00
2013-07-03	12:00:00
2013-09-01	16:00:00
2013-11-27	16:45:00
2013-11-29	12:00:00
2013-12-24	12:00:00

2014-01-19	16:00:00
2014-02-16	16:00:00
2014-05-25	16:00:00
2014-07-03	12:00:00
2014-08-31	16:00:00
2014-11-26	16:45:00
2014-11-28	12:00:00
2014-12-24	12:00:00
2015-01-18	16:00:00
2015-02-15	16:00:00
2015-05-24	16:00:00
2015-07-02	12:00:00
2015-09-06	16:00:00
2015-11-25	16:45:00
2015-11-27	12:05:00
2015-12-24	12:05:00
2016-01-17	16:00:00
2016-02-14	16:00:00
2016-05-29	16:00:00
2016-09-04	16:00:00
2016-11-23	16:45:00
2016-11-25	12:05:00
2016-12-23	12:05:00
2017-01-15	16:00:00
2017-02-19	16:00:00
2017-05-28	16:00:00
2017-07-03	12:05:00
2017-09-03	16:00:00

2017-11-22	16:45:00
2017-12-22	12:05:00
2018-01-14	16:00:00
2018-02-18	16:00:00
2018-05-27	16:00:00
2018-07-03	12:05:00
2018-09-02	16:00:00
2018-11-21	16:45:00
2018-11-23	12:05:00
2018-12-24	12:05:00
2019-01-20	16:00:00
2019-02-17	16:00:00
2019-05-26	16:00:00
2019-07-03	12:05:00
2019-09-01	16:00:00
2019-11-27	16:45:00
2019-11-29	12:05:00
2019-12-24	12:05:00
2020-01-19	16:00:00
2020-02-16	16:00:00
2020-05-24	16:00:00
2020-07-02	12:05:00
2020-09-06	16:00:00
2020-11-25	16:45:00
2020-11-27	12:05:00
2020-12-24	12:05:00
2021-01-17	16:00:00
2021-02-14	16:00:00

2021-05-30	16:00:00
2021-09-05	16:00:00
2021-11-24	16:45:00
2021-11-26	12:05:00
2022-01-16	16:00:00
2022-02-20	16:00:00
2022-05-29	16:00:00
2022-06-19	16:00:00
2022-09-04	16:00:00
2022-11-23	16:45:00
2022-11-25	12:05:00
2023-01-15	16:00:00
2023-02-19	16:00:00
2023-05-28	16:00:00
2023-06-18	16:00:00
2023-09-03	16:00:00
2023-11-22	16:00:00
2023-11-24	12:05:00
2024-03-28	16:00:00
2024-12-24	12:05:00
2024-12-31	16:00:00
2025-01-19	16:00:00
2025-02-16	16:00:00
2025-04-17	16:00:00
2025-05-25	16:00:00
2025-06-18	16:45:00
2025-07-03	16:00:00
2025-08-31	16:00:00

2025-11-26	16:00:00
2025-11-28	12:05:00
2025-12-24	12:05:00
2025-12-31	16:00:00
2026-01-18	16:00:00
2026-02-15	16:00:00
2026-04-02	16:00:00
2026-05-24	16:00:00
2026-06-18	16:00:00
2026-07-02	16:00:00
2026-09-06	16:00:00
2026-11-25	16:00:00
2026-11-27	12:05:00
2026-12-24	12:05:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Chicago SRW Wheat Futures contract in the CBOT Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2011-12-27	09:30:00
2012-01-03	09:30:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-11-23	09:30:00
2013-01-02	09:30:00
2013-01-21	19:00:00
2013-02-18	19:00:00
2013-05-27	19:00:00
2013-07-05	08:30:00

2013-09-02	19:00:00
2013-11-29	08:30:00
2013-12-26	08:30:00
2014-01-02	08:30:00
2014-01-20	19:00:00
2014-02-17	19:00:00
2014-05-26	19:00:00
2014-09-01	19:00:00
2015-01-19	19:00:00
2015-02-16	19:00:00
2015-05-25	19:00:00
2015-09-07	19:00:00
2016-01-18	19:00:00
2016-02-15	19:00:00
2016-05-30	19:00:00
2016-09-05	19:00:00
2016-11-25	08:30:00
2017-01-16	19:00:00
2017-02-20	19:00:00
2017-05-29	19:00:00
2017-07-05	08:30:00
2017-09-04	19:00:00
2017-11-24	08:30:00
2018-01-15	19:00:00
2018-02-19	19:00:00
2018-05-28	19:00:00
2018-07-05	08:30:00
2018-09-03	19:00:00

2018-11-23	08:30:00
2019-01-21	19:00:00
2019-02-18	19:00:00
2019-05-27	19:00:00
2019-07-05	08:30:00
2019-09-02	19:00:00
2019-11-29	08:30:00
2020-01-20	19:00:00
2020-02-17	19:00:00
2020-05-25	19:00:00
2020-09-07	19:00:00
2020-11-27	08:30:00
2021-01-18	19:00:00
2021-02-15	19:00:00
2021-05-31	19:00:00
2021-09-06	19:00:00
2021-11-26	08:30:00
2022-01-17	19:00:00
2022-02-21	19:00:00
2022-05-30	19:00:00
2022-06-20	19:00:00
2022-09-05	19:00:00
2023-01-16	19:00:00
2023-02-20	19:00:00
2023-05-29	19:00:00
2023-06-19	19:00:00
2023-09-04	19:00:00
2024-01-15	19:00:00

2024-02-19	19:00:00
2024-05-27	19:00:00
2024-06-19	19:00:00
2024-09-02	19:00:00
2025-01-20	19:00:00
2025-02-17	19:00:00
2025-05-26	19:00:00
2025-06-19	19:00:00
2025-09-01	19:00:00
2026-01-02	08:30:00
2026-01-19	19:00:00
2026-02-16	19:00:00
2026-05-25	19:00:00
2026-09-07	19:00:00
2026-11-27	08:30:00

Time Zone

The Chicago SRW Wheat Futures contract in the CBOT Future market trades in the `America/Chicago` time zone.

Market Hours

CFE

Introduction

This page shows the trading hours, holidays, and time zone of the CFE Future market.

Pre-market Hours

The following table shows the pre-market hours for the CFE Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the CFE Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

The following table shows the post-market hours for the CFE Future market:

Weekday	Time (America/Chicago)
Monday	15:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	15:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	15:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	15:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	15:00:00 to 16:00:00

Holidays

The following table shows the dates of holidays for the CFE Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02

2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19
2024-07-04	2024-09-02	2024-11-28	2024-12-25	2025-01-01
2025-01-09	2025-01-20	2025-02-17	2025-04-18	2025-05-26
2025-06-19	2025-07-04	2025-09-01	2025-11-27	2025-12-25
2026-01-01	2026-01-19	2026-02-16	2026-04-03	2026-05-25
2026-06-19	2026-07-03	2026-09-07	2026-11-26	2026-12-25

Early Closes

The following table shows the early closes for the CFE Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/Chicago)**
2025-07-03	18:15:00
2025-11-28	18:15:00
2025-12-24	18:15:00
2026-11-27	12:15:00
2026-12-24	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The CFE Future market trades in the `America/Chicago` time zone.

Assets With Other Hours

The following table shows the contracts that have different trading periods than the overall CFE Future market:

Symbol	**Name**
VX	VIX futures

CFE

VX

Introduction

This page shows the trading hours, holidays, and time zone of the VIX futures contract in the CFE Future market.

Pre-market Hours

The following table shows the pre-market hours for the VIX futures contract in the CFE Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the VIX futures contract in the CFE Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

The following table shows the post-market hours for the VIX futures contract in the CFE Future market:

Weekday	Time (America/Chicago)
Monday	15:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	15:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	15:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	15:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	15:00:00 to 16:00:00

Holidays

The following table shows the dates of holidays for the CFE Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02

2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19
2024-07-04	2024-09-02	2024-11-28	2024-12-25	2025-01-01
2025-01-09	2025-01-20	2025-02-17	2025-04-18	2025-05-26
2025-06-19	2025-07-04	2025-09-01	2025-11-27	2025-12-25
2026-01-01	2026-01-19	2026-02-16	2026-04-03	2026-05-25
2026-06-19	2026-07-03	2026-09-07	2026-11-26	2026-12-25

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VIX futures contract in the CFE Future market trades in the `America/Chicago` time zone.

Market Hours

CME

Introduction

This page shows the trading hours, holidays, and time zone of the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the CME Future market:

Weekday	Time (America/Chicago)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04

2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-04-10
2009-12-25	2010-01-01	2010-12-24	2011-01-01	2011-04-22
2011-12-26	2012-01-02	2012-10-29	2012-10-30	2012-12-25
2013-01-01	2013-03-29	2013-12-25	2014-01-01	2014-04-18
2014-12-25	2015-01-01	2015-12-25	2016-01-01	2016-03-25
2016-12-25	2016-12-26	2017-01-02	2017-04-14	2017-12-25
2018-01-01	2018-03-30	2018-12-05	2018-12-25	2019-01-01
2019-04-19	2019-12-25	2020-01-01	2020-04-10	2020-12-25
2021-01-01	2021-12-24	2022-01-01	2022-04-15	2022-12-26
2023-01-02	2023-12-24	2023-12-31	2024-03-29	2025-01-20
2025-04-18				

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CME Future market trades in the following time zones:

- `America/Chicago`
- `America/New York`

Assets With Other Hours

The following table shows the contracts that have different trading periods than the overall CME Future market:

Symbol	Name
6A	Australian Dollar Futures
6B	British Pound Futures
6C	Canadian Dollar Futures
6E	Euro FX Futures
6J	Japanese Yen Futures
6L	Brazilian Real Futures
6M	Mexican Peso Futures

6N	New Zealand Dollar Futures
6R	Russian Ruble Futures
6S	Swiss Franc Futures
6Z	South African Rand Futures
ACD	Australian Dollar/Canadian Dollar Futures
AJY	Australian Dollar/Japanese Yen Futures
ANE	Australian Dollar/New Zealand Dollar Futures
BIO	E-mini Nasdaq-100 Biotechnology Index Futures
BTC	Bitcoin Futures
CB	CB
CJY	Canadian Dollar/Japanese Yen Futures
CNH	Standard-Size USD/Offshore RMB (CNH) Futures
CSC	CSC
DC	DC
DY	DY
E3G	E-mini FTSE 100 GBP Futures
E7	E-mini Euro FX Futures
EAD	Euro/Australian Dollar Futures
ECD	Euro/Canadian Dollar Futures
EI	E-mini FTSE Emerging Index Futures
EMD	E-mini S&P MidCap 400 Futures
ENY	E-mini Nikkei YEN Futures
ES	E-mini S&P 500 Futures
ESG	E-mini S&P500 ESG Futures
ESK	Euro/Swedish Krona Futures
ETH	Ether Futures
FT1	E-mini FTSE 100 GBP Futures
FT5	E-mini FTSE China Futures

FTU	E-mini FTSE 100 USD Futures
GD	S&P-GSCI Commodity Index Futures
GDK	GDK
GE	Eurodollar Futures
GF	Feeder Cattle Futures
GNF	GNF
HE	Lean Hog Futures
IBV	USD-Denominated Ibovespa Index Futures
J7	E-mini Japanese Yen Futures
LBR	Lumber Futures
LBS	Random Length Lumber Futures
LE	Live Cattle Futures
M2K	Micro E-mini Russell 2000 Index Futures
M6A	Micro Australian Dollar/U.S. Dollar (AUD/USD) Futures
M6B	Micro British Pound Sterling/U.S. Dollar (GBP/USD) Futures
M6C	Micro USD/CAD Futures
M6E	Micro Euro/U.S. Dollar (EUR/USD) Futures
M6J	Micro USD/JPY Futures
M6S	Micro USD/CHF Futures
MBT	Micro Bitcoin Futures
MCD	Micro Canadian Dollar/U.S.Dollar(CAD/USD) Futures
MES	Micro E-mini Standard and Poor's 500 Stock Price Index Futures
MET	Micro Ether Futures
MIB	BTIC on Micro Bitcoin Futures
MIR	Micro INR/USD Futures
MJY	Micro Japanese Yen/U.S. Dollar (JPY/USD) Futures
MNH	Micro USD/CNH Futures
MNQ	Micro E-mini Nasdaq-100 Index Futures

MRB	BTIC on Micro Ether Futures
MSF	Micro Swiss Franc/U.S. Dollar (CHF/USD) Futures
NIY	Nikkei/YEN Futures
NKD	Nikkei/USD Futures
NQ	E-mini Nasdaq-100 Futures
RS1	E-mini Russell 1000 future
RTY	E-mini Russell 2000 Index Futures
RX	DJRE Futures
SDA	SP500 S&P 500 Annual Dividend Index future
TPD	TOPIX USD Futures
TPY	TOPIX JPY Futures

CME

6A

Introduction

This page shows the trading hours, holidays, and time zone of the Australian Dollar Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Australian Dollar Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Australian Dollar Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Australian Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Australian Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Australian Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00
2024-01-15	16:00:00

2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00

2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Australian Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00

2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Australian Dollar Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

6B

Introduction

This page shows the trading hours, holidays, and time zone of the British Pound Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the British Pound Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the British Pound Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the British Pound Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the British Pound Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the British Pound Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00
2024-01-15	16:00:00

2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00

2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the British Pound Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00

2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The British Pound Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

6C

Introduction

This page shows the trading hours, holidays, and time zone of the Canadian Dollar Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Canadian Dollar Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Canadian Dollar Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Canadian Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Canadian Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Canadian Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00
2024-01-15	16:00:00

2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00

2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Canadian Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00

2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Canadian Dollar Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

6E

Introduction

This page shows the trading hours, holidays, and time zone of the Euro FX Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Euro FX Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Euro FX Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Euro FX Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Euro FX Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Euro FX Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00
2024-01-15	16:00:00

2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00

2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Euro FX Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00

2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Euro FX Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

6J

Introduction

This page shows the trading hours, holidays, and time zone of the Japanese Yen Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Japanese Yen Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Japanese Yen Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Japanese Yen Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Japanese Yen Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Japanese Yen Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00
2024-01-15	16:00:00

2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00

2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Japanese Yen Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00

2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Japanese Yen Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

6L

Introduction

This page shows the trading hours, holidays, and time zone of the Brazilian Real Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Brazilian Real Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Brazilian Real Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Brazilian Real Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Brazilian Real Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Brazilian Real Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00
2024-01-15	16:00:00

2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00

2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Brazilian Real Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00

2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Brazilian Real Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

6M

Introduction

This page shows the trading hours, holidays, and time zone of the Mexican Peso Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mexican Peso Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mexican Peso Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Mexican Peso Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Mexican Peso Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Mexican Peso Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00
2024-01-15	16:00:00

2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00

2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Mexican Peso Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00

2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Mexican Peso Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

6N

Introduction

This page shows the trading hours, holidays, and time zone of the New Zealand Dollar Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the New Zealand Dollar Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the New Zealand Dollar Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the New Zealand Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the New Zealand Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the New Zealand Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00
2024-01-15	16:00:00

2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00

2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the New Zealand Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00

2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The New Zealand Dollar Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

6R

Introduction

This page shows the trading hours, holidays, and time zone of the Russian Ruble Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Russian Ruble Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Russian Ruble Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Russian Ruble Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Russian Ruble Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Russian Ruble Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00
2024-01-15	16:00:00

2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00

2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Russian Ruble Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00

2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Russian Ruble Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

6S

Introduction

This page shows the trading hours, holidays, and time zone of the Swiss Franc Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Swiss Franc Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Swiss Franc Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Swiss Franc Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Swiss Franc Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Swiss Franc Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00
2024-01-15	16:00:00

2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00

2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Swiss Franc Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00

2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Swiss Franc Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

6Z

Introduction

This page shows the trading hours, holidays, and time zone of the South African Rand Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the South African Rand Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the South African Rand Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the South African Rand Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the South African Rand Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the South African Rand Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00
2024-01-15	16:00:00

2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00

2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the South African Rand Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00

2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The South African Rand Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

ACD

Introduction

This page shows the trading hours, holidays, and time zone of the Australian Dollar/Canadian Dollar Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Australian Dollar/Canadian Dollar Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Australian Dollar/Canadian Dollar Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Australian Dollar/Canadian Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Australian Dollar/Canadian Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Australian Dollar/Canadian Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00

2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00

2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00

2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00

2024-01-15	16:00:00
2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00

2026-11-26	16:00:00
2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Australian Dollar/Canadian Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00

2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00

2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00

2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00

2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Australian Dollar/Canadian Dollar Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

AJY

Introduction

This page shows the trading hours, holidays, and time zone of the Australian Dollar/Japanese Yen Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Australian Dollar/Japanese Yen Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Australian Dollar/Japanese Yen Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Australian Dollar/Japanese Yen Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Australian Dollar/Japanese Yen Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Australian Dollar/Japanese Yen Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00

2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00

2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00

2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00

2024-01-15	16:00:00
2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00

2026-11-26	16:00:00
2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Australian Dollar/Japanese Yen Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00

2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00

2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00

2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00

2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Australian Dollar/Japanese Yen Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

ANE

Introduction

This page shows the trading hours, holidays, and time zone of the Australian Dollar/New Zealand Dollar Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Australian Dollar/New Zealand Dollar Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Australian Dollar/New Zealand Dollar Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Australian Dollar/New Zealand Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Australian Dollar/New Zealand Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Australian Dollar/New Zealand Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00

2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00

2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00

2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00

2024-01-15	16:00:00
2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00

2026-11-26	16:00:00
2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Australian Dollar/New Zealand Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00

2012-07-04	17:00:00
2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00

2017-07-04	17:00:00
2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00

2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00

2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Australian Dollar/New Zealand Dollar Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

BIO

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini Nasdaq-100 Biotechnology Index Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini Nasdaq-100 Biotechnology Index Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini Nasdaq-100 Biotechnology Index Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini Nasdaq-100 Biotechnology Index Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the E-mini Nasdaq-100 Biotechnology Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the E-mini Nasdaq-100 Biotechnology Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-04-02	08:15:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00

2010-11-26	12:15:00
2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-04-06	08:15:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-20	10:30:00

2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:15:00
2014-07-04	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00

2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	08:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00
2023-01-16	12:00:00
2023-02-20	12:00:00
2023-04-07	08:15:00
2023-05-29	12:00:00
2023-06-19	12:00:00
2023-07-04	12:00:00
2023-09-04	12:00:00
2023-11-23	12:00:00
2023-11-24	12:15:00
2024-01-15	12:00:00

2024-02-19	12:00:00
2024-05-27	12:00:00
2024-06-19	12:00:00
2024-07-04	12:00:00
2024-09-02	12:00:00
2024-11-28	12:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	12:00:00
2025-02-17	12:00:00
2025-04-17	16:00:00
2025-05-26	12:00:00
2025-06-19	12:00:00
2025-07-03	12:15:00
2025-07-04	12:00:00
2025-09-01	12:00:00
2025-11-27	12:00:00
2025-11-28	12:15:00
2025-12-24	12:15:00
2025-12-25	16:00:00
2025-12-31	16:00:00
2026-01-19	12:00:00
2026-02-16	12:00:00
2026-04-03	08:15:00
2026-05-25	12:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	12:00:00

2026-11-26	12:00:00
2026-11-27	12:15:00
2026-12-24	12:15:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the E-mini Nasdaq-100 Biotechnology Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00

2012-07-04	17:00:00
2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00

2017-07-04	17:00:00
2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00

2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-07-03	17:00:00

2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The E-mini Nasdaq-100 Biotechnology Index Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

BTC

Introduction

This page shows the trading hours, holidays, and time zone of the Bitcoin Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Bitcoin Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Bitcoin Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Bitcoin Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Bitcoin Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Bitcoin Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2018-01-15	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00

2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	08:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:45:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:45:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00

2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:45:00
2024-01-15	16:00:00
2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:45:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00

2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00
2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Bitcoin Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2018-01-15	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00

2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00

2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Bitcoin Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

CB

Introduction

This page shows the trading hours, holidays, and time zone of the CB contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the CB contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the CB contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the CB contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CB contract in the CME Future market:

Date (*yyyy-mm-dd*)				
2014-07-04	2015-04-03	2015-07-03	2017-11-23	2018-11-22
2019-11-28	2020-07-03	2020-11-26	2021-04-02	2021-11-25
2022-11-24	2023-04-07	2023-11-23	2024-11-28	2025-07-04
2025-11-27	2025-11-28	2026-04-03	2026-06-19	2026-07-03
2026-11-26	2026-11-27	2026-12-25	2027-01-01	

Early Closes

The following table shows the early closes for the CB contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-11-25	16:00:00
2010-12-31	12:15:00
2011-04-21	13:55:00
2011-11-23	16:00:00
2012-01-15	17:00:00
2012-02-19	17:00:00
2012-05-27	17:00:00
2012-07-03	16:00:00
2012-09-02	17:00:00
2012-11-21	16:00:00
2012-12-24	12:00:00
2013-01-20	17:00:00

2013-02-17	17:00:00
2013-03-28	13:55:00
2013-05-26	17:00:00
2013-07-03	12:00:00
2013-09-01	17:00:00
2013-11-27	17:00:00
2013-12-24	12:00:00
2014-02-16	16:00:00
2014-04-17	13:55:00
2014-05-25	16:00:00
2014-07-03	12:00:00
2014-12-24	12:00:00
2015-04-02	13:55:00
2015-07-02	12:00:00
2015-12-24	12:00:00
2015-12-31	13:55:00
2016-03-24	13:55:00
2016-12-23	12:00:00
2017-01-15	16:00:00
2017-02-19	16:00:00
2017-04-13	13:55:00
2017-05-28	16:00:00
2017-07-03	12:00:00
2017-09-03	16:00:00
2017-12-22	12:00:00
2018-01-14	16:00:00
2018-02-18	16:00:00
2018-03-29	13:55:00

2018-05-27	16:00:00
2018-07-03	12:00:00
2018-09-02	16:00:00
2018-12-24	12:00:00
2018-12-31	13:55:00
2019-01-20	16:00:00
2019-02-17	16:00:00
2019-04-18	13:55:00
2019-05-26	16:00:00
2019-07-03	12:00:00
2019-09-01	16:00:00
2019-11-27	16:00:00
2019-12-24	12:00:00
2019-12-31	13:55:00
2020-01-19	16:00:00
2020-02-16	16:00:00
2020-04-09	13:55:00
2020-05-24	16:00:00
2020-07-02	12:00:00
2020-09-06	16:00:00
2020-11-25	16:45:00
2020-12-24	12:00:00
2020-12-31	13:55:00
2021-01-17	16:00:00
2021-02-14	16:00:00
2021-04-01	13:55:00
2021-05-30	16:00:00
2021-07-04	16:00:00

2021-09-05	16:00:00
2022-01-16	16:00:00
2022-02-20	16:00:00
2022-04-14	13:55:00
2022-05-29	16:00:00
2022-06-19	16:00:00
2022-07-03	16:00:00
2022-09-04	16:00:00
2023-01-15	16:00:00
2023-02-19	16:00:00
2023-04-06	13:55:00
2023-05-28	16:00:00
2023-06-18	16:00:00
2023-09-03	16:00:00
2023-11-22	16:00:00
2024-01-01	17:00:00
2024-03-28	13:55:00
2024-11-27	16:00:00
2024-12-24	12:00:00
2024-12-31	16:00:00
2025-01-19	16:00:00
2025-02-16	16:00:00
2025-04-17	13:55:00
2025-05-25	16:00:00
2025-07-03	12:00:00
2025-08-31	16:00:00
2025-11-26	16:00:00
2025-12-24	12:00:00

2025-12-31	16:00:00
2026-01-02	13:55:00
2026-01-18	16:00:00
2026-02-15	16:00:00
2026-04-02	13:55:00
2026-05-24	16:00:00
2026-06-18	13:55:00
2026-07-02	13:55:00
2026-09-06	16:00:00
2026-11-25	16:00:00
2026-12-24	12:00:00
2026-12-31	13:55:00

Late Opens

The following table shows the late opens for the CB contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2011-12-27	09:05:00
2012-01-03	09:05:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00
2012-12-26	09:05:00
2013-01-02	09:05:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00

2013-09-02	17:00:00
2013-12-26	09:05:00
2014-01-02	09:05:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-12-26	09:05:00
2015-01-02	09:05:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-03-27	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-29	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-05	08:30:00
2017-09-04	17:00:00
2017-11-26	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00

2018-05-28	17:00:00
2018-09-03	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-04-18	13:55:00
2019-05-27	17:00:00
2019-09-02	17:00:00
2019-12-01	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-29	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00

2023-09-04	17:00:00
2023-12-25	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00

Time Zone

The CB contract in the CME Future market trades in the `America/Chicago` time zone.

CME

CJY

Introduction

This page shows the trading hours, holidays, and time zone of the Canadian Dollar/Japanese Yen Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Canadian Dollar/Japanese Yen Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Canadian Dollar/Japanese Yen Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Canadian Dollar/Japanese Yen Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Canadian Dollar/Japanese Yen Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Canadian Dollar/Japanese Yen Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00

2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00

2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00

2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00

2024-01-15	16:00:00
2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00

2026-11-26	16:00:00
2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Canadian Dollar/Japanese Yen Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00

2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00

2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00

2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00

2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Canadian Dollar/Japanese Yen Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

CNH

Introduction

This page shows the trading hours, holidays, and time zone of the Standard-Size USD/Offshore RMB (CNH) Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Standard-Size USD/Offshore RMB (CNH) Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Standard-Size USD/Offshore RMB (CNH) Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Standard-Size USD/Offshore RMB (CNH) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Standard-Size USD/Offshore RMB (CNH) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Standard-Size USD/Offshore RMB (CNH) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00

2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00

2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00

2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00

2024-01-15	16:00:00
2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00

2026-11-26	16:00:00
2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Standard-Size USD/Offshore RMB (CNH) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00

2012-07-04	17:00:00
2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00

2017-07-04	17:00:00
2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00

2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00

2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Standard-Size USD/Offshore RMB (CNH) Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

CSC

Introduction

This page shows the trading hours, holidays, and time zone of the CSC contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the CSC contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the CSC contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the CSC contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CSC contract in the CME Future market:

Date (*yyyy-mm-dd*)				
2014-07-04	2015-04-03	2015-07-03	2017-11-23	2018-11-22
2019-11-28	2020-07-03	2020-11-26	2021-04-02	2021-11-25
2022-11-24	2023-04-07	2023-11-23	2024-11-28	2025-07-04
2025-11-27	2025-11-28	2026-04-03	2026-06-19	2026-07-03
2026-11-26	2026-11-27	2026-12-25	2027-01-01	

Early Closes

The following table shows the early closes for the CSC contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-11-25	16:00:00
2010-12-31	12:15:00
2011-04-21	13:55:00
2011-11-23	16:00:00
2012-01-15	17:00:00
2012-02-19	17:00:00
2012-05-27	17:00:00
2012-07-03	16:00:00
2012-09-02	17:00:00
2012-11-21	16:00:00
2012-12-24	12:00:00
2013-01-20	17:00:00

2013-02-17	17:00:00
2013-03-28	13:55:00
2013-05-26	17:00:00
2013-07-03	12:00:00
2013-09-01	17:00:00
2013-11-27	17:00:00
2013-12-24	12:00:00
2014-02-16	16:00:00
2014-04-17	13:55:00
2014-05-25	16:00:00
2014-07-03	12:00:00
2014-12-24	12:00:00
2015-04-02	13:55:00
2015-07-02	12:00:00
2015-12-24	12:00:00
2015-12-31	13:55:00
2016-03-24	13:55:00
2016-12-23	12:00:00
2017-01-15	16:00:00
2017-02-19	16:00:00
2017-04-13	13:55:00
2017-05-28	16:00:00
2017-07-03	12:00:00
2017-09-03	16:00:00
2017-12-22	12:00:00
2018-01-14	16:00:00
2018-02-18	16:00:00
2018-03-29	13:55:00

2018-05-27	16:00:00
2018-07-03	12:00:00
2018-09-02	16:00:00
2018-12-24	12:00:00
2018-12-31	13:55:00
2019-01-20	16:00:00
2019-02-17	16:00:00
2019-04-18	13:55:00
2019-05-26	16:00:00
2019-07-03	12:00:00
2019-09-01	16:00:00
2019-11-27	16:00:00
2019-12-24	12:00:00
2019-12-31	13:55:00
2020-01-19	16:00:00
2020-02-16	16:00:00
2020-04-09	13:55:00
2020-05-24	16:00:00
2020-07-02	12:00:00
2020-09-06	16:00:00
2020-11-25	16:45:00
2020-12-24	12:00:00
2020-12-31	13:55:00
2021-01-17	16:00:00
2021-02-14	16:00:00
2021-04-01	13:55:00
2021-05-30	16:00:00
2021-07-04	16:00:00

2021-09-05	16:00:00
2022-01-16	16:00:00
2022-02-20	16:00:00
2022-04-14	13:55:00
2022-05-29	16:00:00
2022-06-19	16:00:00
2022-07-03	16:00:00
2022-09-04	16:00:00
2023-01-15	16:00:00
2023-02-19	16:00:00
2023-04-06	13:55:00
2023-05-28	16:00:00
2023-06-18	16:00:00
2023-09-03	16:00:00
2023-11-22	16:00:00
2024-01-01	17:00:00
2024-03-28	13:55:00
2024-11-27	16:00:00
2024-12-24	12:00:00
2024-12-31	16:00:00
2025-01-19	16:00:00
2025-02-16	16:00:00
2025-04-17	13:55:00
2025-05-25	16:00:00
2025-07-03	12:00:00
2025-08-31	16:00:00
2025-11-26	16:00:00
2025-12-24	12:00:00

2025-12-31	16:00:00
2026-01-02	13:55:00
2026-01-18	16:00:00
2026-02-15	16:00:00
2026-04-02	13:55:00
2026-05-24	16:00:00
2026-06-18	13:55:00
2026-07-02	13:55:00
2026-09-06	16:00:00
2026-11-25	16:00:00
2026-12-24	12:00:00
2026-12-31	13:55:00

Late Opens

The following table shows the late opens for the CSC contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2011-12-27	09:05:00
2012-01-03	09:05:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00
2012-12-26	09:05:00
2013-01-02	09:05:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00

2013-09-02	17:00:00
2013-12-26	09:05:00
2014-01-02	09:05:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-12-26	09:05:00
2015-01-02	09:05:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-03-27	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-29	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-05	08:30:00
2017-09-04	17:00:00
2017-11-26	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00

2018-05-28	17:00:00
2018-09-03	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-04-18	13:55:00
2019-05-27	17:00:00
2019-09-02	17:00:00
2019-12-01	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-29	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00

2023-09-04	17:00:00
2023-12-25	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00

Time Zone

The CSC contract in the CME Future market trades in the `America/Chicago` time zone.

CME

DC

Introduction

This page shows the trading hours, holidays, and time zone of the DC contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the DC contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the DC contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the DC contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the DC contract in the CME Future market:

Date (*yyyy-mm-dd*)				
2014-07-04	2015-04-03	2015-07-03	2017-11-23	2018-11-22
2019-11-28	2020-07-03	2020-11-26	2021-04-02	2021-11-25
2022-11-24	2023-04-07	2023-11-23	2024-11-28	2025-07-04
2025-11-27	2025-11-28	2026-04-03	2026-06-19	2026-07-03
2026-11-26	2026-11-27	2026-12-25	2027-01-01	

Early Closes

The following table shows the early closes for the DC contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-11-25	16:00:00
2010-12-31	12:15:00
2011-04-21	13:55:00
2011-11-23	16:00:00
2012-01-15	17:00:00
2012-02-19	17:00:00
2012-05-27	17:00:00
2012-07-03	16:00:00
2012-09-02	17:00:00
2012-11-21	16:00:00
2012-12-24	12:00:00
2013-01-20	17:00:00

2013-02-17	17:00:00
2013-03-28	13:55:00
2013-05-26	17:00:00
2013-07-03	12:00:00
2013-09-01	17:00:00
2013-11-27	17:00:00
2013-12-24	12:00:00
2014-02-16	16:00:00
2014-04-17	13:55:00
2014-05-25	16:00:00
2014-07-03	12:00:00
2014-12-24	12:00:00
2015-04-02	13:55:00
2015-07-02	12:00:00
2015-12-24	12:00:00
2015-12-31	13:55:00
2016-03-24	13:55:00
2016-12-23	12:00:00
2017-01-15	16:00:00
2017-02-19	16:00:00
2017-04-13	13:55:00
2017-05-28	16:00:00
2017-07-03	12:00:00
2017-09-03	16:00:00
2017-12-22	12:00:00
2018-01-14	16:00:00
2018-02-18	16:00:00
2018-03-29	13:55:00

2018-05-27	16:00:00
2018-07-03	12:00:00
2018-09-02	16:00:00
2018-12-24	12:00:00
2018-12-31	13:55:00
2019-01-20	16:00:00
2019-02-17	16:00:00
2019-04-18	13:55:00
2019-05-26	16:00:00
2019-07-03	12:00:00
2019-09-01	16:00:00
2019-11-27	16:00:00
2019-12-24	12:00:00
2019-12-31	13:55:00
2020-01-19	16:00:00
2020-02-16	16:00:00
2020-04-09	13:55:00
2020-05-24	16:00:00
2020-07-02	12:00:00
2020-09-06	16:00:00
2020-11-25	16:45:00
2020-12-24	12:00:00
2020-12-31	13:55:00
2021-01-17	16:00:00
2021-02-14	16:00:00
2021-04-01	13:55:00
2021-05-30	16:00:00
2021-07-04	16:00:00

2021-09-05	16:00:00
2022-01-16	16:00:00
2022-02-20	16:00:00
2022-04-14	13:55:00
2022-05-29	16:00:00
2022-06-19	16:00:00
2022-07-03	16:00:00
2022-09-04	16:00:00
2023-01-15	16:00:00
2023-02-19	16:00:00
2023-04-06	13:55:00
2023-05-28	16:00:00
2023-06-18	16:00:00
2023-09-03	16:00:00
2023-11-22	16:00:00
2024-01-01	17:00:00
2024-03-28	13:55:00
2024-11-27	16:00:00
2024-12-24	12:00:00
2024-12-31	16:00:00
2025-01-19	16:00:00
2025-02-16	16:00:00
2025-04-17	13:55:00
2025-05-25	16:00:00
2025-07-03	12:00:00
2025-08-31	16:00:00
2025-11-26	16:00:00
2025-12-24	12:00:00

2025-12-31	16:00:00
2026-01-02	13:55:00
2026-01-18	16:00:00
2026-02-15	16:00:00
2026-04-02	13:55:00
2026-05-24	16:00:00
2026-06-18	13:55:00
2026-07-02	13:55:00
2026-09-06	16:00:00
2026-11-25	16:00:00
2026-12-24	12:00:00
2026-12-31	13:55:00

Late Opens

The following table shows the late opens for the DC contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2011-12-27	09:05:00
2012-01-03	09:05:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00
2012-12-26	09:05:00
2013-01-02	09:05:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00

2013-09-02	17:00:00
2013-12-26	09:05:00
2014-01-02	09:05:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-12-26	09:05:00
2015-01-02	09:05:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-03-27	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-29	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-05	08:30:00
2017-09-04	17:00:00
2017-11-26	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00

2018-05-28	17:00:00
2018-09-03	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-04-18	13:55:00
2019-05-27	17:00:00
2019-09-02	17:00:00
2019-12-01	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-29	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00

2023-09-04	17:00:00
2023-12-25	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00

Time Zone

The DC contract in the CME Future market trades in the `America/Chicago` time zone.

CME

DY

Introduction

This page shows the trading hours, holidays, and time zone of the DY contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the DY contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the DY contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 13:55:00

Post-market Hours

The following table shows the post-market hours for the DY contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the DY contract in the CME Future market:

Date (*yyyy-mm-dd*)				
2014-07-04	2015-04-03	2015-07-03	2017-11-23	2018-11-22
2019-11-28	2020-07-03	2020-11-26	2021-04-02	2021-11-25
2022-11-24	2023-04-07	2023-11-23	2024-11-28	2025-07-04
2025-11-27	2025-11-28	2026-04-03	2026-06-19	2026-07-03
2026-11-26	2026-11-27	2026-12-25	2027-01-01	

Early Closes

The following table shows the early closes for the DY contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-11-25	16:00:00
2010-12-31	12:15:00
2011-04-21	13:55:00
2011-11-23	16:00:00
2012-01-15	17:00:00
2012-02-19	17:00:00
2012-05-27	17:00:00
2012-07-03	16:00:00
2012-09-02	17:00:00
2012-11-21	16:00:00
2012-12-24	12:00:00
2013-01-20	17:00:00

2013-02-17	17:00:00
2013-03-28	13:55:00
2013-05-26	17:00:00
2013-07-03	12:00:00
2013-09-01	17:00:00
2013-11-27	17:00:00
2013-12-24	12:00:00
2014-02-16	16:00:00
2014-04-17	13:55:00
2014-05-25	16:00:00
2014-07-03	12:00:00
2014-12-24	12:00:00
2015-04-02	13:55:00
2015-07-02	12:00:00
2015-12-24	12:00:00
2015-12-31	13:55:00
2016-03-24	13:55:00
2016-12-23	12:00:00
2017-01-15	16:00:00
2017-02-19	16:00:00
2017-04-13	13:55:00
2017-05-28	16:00:00
2017-07-03	12:00:00
2017-09-03	16:00:00
2017-12-22	12:00:00
2018-01-14	16:00:00
2018-02-18	16:00:00
2018-03-29	13:55:00

2018-05-27	16:00:00
2018-07-03	12:00:00
2018-09-02	16:00:00
2018-12-24	12:00:00
2018-12-31	13:55:00
2019-01-20	16:00:00
2019-02-17	16:00:00
2019-04-18	13:55:00
2019-05-26	16:00:00
2019-07-03	12:00:00
2019-09-01	16:00:00
2019-11-27	16:00:00
2019-12-24	12:00:00
2019-12-31	13:55:00
2020-01-19	16:00:00
2020-02-16	16:00:00
2020-04-09	13:55:00
2020-05-24	16:00:00
2020-07-02	12:00:00
2020-09-06	16:00:00
2020-11-25	16:45:00
2020-12-24	12:00:00
2020-12-31	13:55:00
2021-01-17	16:00:00
2021-02-14	16:00:00
2021-04-01	13:55:00
2021-05-30	16:00:00
2021-07-04	16:00:00

2021-09-05	16:00:00
2022-01-16	16:00:00
2022-02-20	16:00:00
2022-04-14	13:55:00
2022-05-29	16:00:00
2022-06-19	16:00:00
2022-07-03	16:00:00
2022-09-04	16:00:00
2023-01-15	16:00:00
2023-02-19	16:00:00
2023-04-06	13:55:00
2023-05-28	16:00:00
2023-06-18	16:00:00
2023-09-03	16:00:00
2023-11-22	16:00:00
2024-01-01	17:00:00
2024-03-28	13:55:00
2024-11-27	16:00:00
2024-12-24	12:00:00
2024-12-31	16:00:00
2025-01-19	16:00:00
2025-02-16	16:00:00
2025-04-17	13:55:00
2025-05-25	16:00:00
2025-07-03	12:00:00
2025-08-31	16:00:00
2025-11-26	16:00:00
2025-12-24	12:00:00

2025-12-31	16:00:00
2026-01-02	13:55:00
2026-01-18	16:00:00
2026-02-15	16:00:00
2026-04-02	13:55:00
2026-05-24	16:00:00
2026-06-18	13:55:00
2026-07-02	13:55:00
2026-09-06	16:00:00
2026-11-25	16:00:00
2026-12-24	12:00:00
2026-12-31	13:55:00

Late Opens

The following table shows the late opens for the DY contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2011-12-27	09:05:00
2012-01-03	09:05:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00
2012-12-26	09:05:00
2013-01-02	09:05:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00

2013-09-02	17:00:00
2013-12-26	09:05:00
2014-01-02	09:05:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-12-26	09:05:00
2015-01-02	09:05:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-03-27	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-29	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-05	08:30:00
2017-09-04	17:00:00
2017-11-26	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00

2018-05-28	17:00:00
2018-09-03	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-04-18	13:55:00
2019-05-27	17:00:00
2019-09-02	17:00:00
2019-12-01	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-29	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00

2023-09-04	17:00:00
2023-12-25	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00

Time Zone

The DY contract in the CME Future market trades in the `America/Chicago` time zone.

CME

E3G

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini FTSE 100 GBP Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini FTSE 100 GBP Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini FTSE 100 GBP Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini FTSE 100 GBP Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the E-mini FTSE 100 GBP Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the E-mini FTSE 100 GBP Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	11:30:00
2009-02-16	11:30:00
2009-05-25	11:30:00
2009-07-03	11:30:00
2009-09-07	11:30:00
2009-11-26	11:30:00
2009-11-27	13:15:00
2009-12-24	13:15:00
2010-01-18	11:30:00
2010-02-15	11:30:00
2010-04-02	09:15:00
2010-05-31	11:30:00
2010-07-05	11:30:00
2010-09-06	11:30:00
2010-11-25	11:30:00
2010-11-26	13:15:00

2011-01-17	11:30:00
2011-02-21	11:30:00
2011-05-30	11:30:00
2011-07-04	11:30:00
2011-09-05	11:30:00
2011-11-24	11:30:00
2011-11-25	13:15:00
2012-01-16	11:30:00
2012-02-20	11:30:00
2012-04-06	09:15:00
2012-05-28	11:30:00
2012-07-03	13:15:00
2012-07-04	11:30:00
2012-09-03	11:30:00
2012-11-22	11:30:00
2012-11-23	13:15:00
2012-12-24	13:15:00
2013-01-21	11:30:00
2013-02-18	11:30:00
2013-05-27	11:30:00
2013-07-03	13:15:00
2013-07-04	11:30:00
2013-09-02	11:30:00
2013-11-28	11:30:00
2013-11-29	13:15:00
2013-12-24	13:15:00
2014-01-20	11:30:00
2014-02-17	11:30:00

2014-05-26	13:00:00
2014-07-03	13:15:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:15:00
2014-12-24	13:15:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-04-03	09:15:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:15:00
2015-12-24	13:15:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:15:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-03	13:15:00
2017-07-04	13:00:00

2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:15:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-03	13:15:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:15:00
2018-12-24	13:15:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-03	13:15:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:15:00
2019-12-24	13:15:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:15:00

2020-12-24	13:15:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-04-02	09:15:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:15:00
2022-01-17	13:00:00
2022-02-21	13:00:00
2022-05-30	13:00:00
2022-06-20	13:00:00
2022-07-04	13:00:00
2022-09-05	13:00:00
2022-11-24	13:00:00
2022-11-25	13:15:00
2023-01-16	13:00:00
2023-02-20	13:00:00
2023-04-07	09:15:00
2023-05-29	13:00:00
2023-06-19	13:00:00
2023-07-04	13:00:00
2023-09-04	13:00:00
2023-11-23	13:00:00
2023-11-24	13:15:00
2024-01-15	13:00:00
2024-02-19	13:00:00

2024-05-27	13:00:00
2024-06-19	13:00:00
2024-07-04	13:00:00
2024-09-02	13:00:00
2024-11-28	13:00:00
2024-12-24	13:15:00
2024-12-31	17:00:00
2025-01-20	13:00:00
2025-02-17	13:00:00
2025-04-17	17:00:00
2025-05-26	13:00:00
2025-06-19	13:00:00
2025-07-03	13:15:00
2025-07-04	13:00:00
2025-09-01	13:00:00
2025-11-27	13:00:00
2025-11-28	13:15:00
2025-12-24	13:15:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	13:00:00
2026-02-16	13:00:00
2026-04-03	09:15:00
2026-05-25	13:00:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	13:00:00
2026-11-26	13:00:00

2026-11-27	13:15:00
2026-12-24	13:15:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the E-mini FTSE 100 GBP Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00

2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-07-03	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The E-mini FTSE 100 GBP Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

E7

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini Euro FX Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini Euro FX Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini Euro FX Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini Euro FX Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the E-mini Euro FX Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the E-mini Euro FX Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00
2024-01-15	16:00:00

2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00

2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the E-mini Euro FX Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00

2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The E-mini Euro FX Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

EAD

Introduction

This page shows the trading hours, holidays, and time zone of the Euro/Australian Dollar Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Euro/Australian Dollar Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Euro/Australian Dollar Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Euro/Australian Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Euro/Australian Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Euro/Australian Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00
2024-01-15	16:00:00

2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00

2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Euro/Australian Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00

2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Euro/Australian Dollar Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

ECD

Introduction

This page shows the trading hours, holidays, and time zone of the Euro/Canadian Dollar Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Euro/Canadian Dollar Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Euro/Canadian Dollar Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Euro/Canadian Dollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Euro/Canadian Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Euro/Canadian Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00
2024-01-15	16:00:00

2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00

2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Euro/Canadian Dollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00

2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Euro/Canadian Dollar Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

EI

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini FTSE Emerging Index Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini FTSE Emerging Index Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini FTSE Emerging Index Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini FTSE Emerging Index Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the E-mini FTSE Emerging Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the E-mini FTSE Emerging Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-04-02	08:15:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00
2010-11-26	12:15:00

2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-04-06	08:15:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-20	10:30:00
2014-02-17	10:30:00

2014-05-26	12:00:00
2014-07-03	12:15:00
2014-07-04	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00
2017-07-04	12:00:00

2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00

2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	08:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00
2023-01-16	12:00:00
2023-02-20	12:00:00
2023-04-07	08:15:00
2023-05-29	12:00:00
2023-06-19	12:00:00
2023-07-04	12:00:00
2023-09-04	12:00:00
2023-11-23	12:00:00
2023-11-24	12:15:00
2024-01-15	12:00:00
2024-02-19	12:00:00

2024-05-27	12:00:00
2024-06-19	12:00:00
2024-07-04	12:00:00
2024-09-02	12:00:00
2024-11-28	12:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	12:00:00
2025-02-17	12:00:00
2025-04-17	16:00:00
2025-05-26	12:00:00
2025-06-19	12:00:00
2025-07-03	12:15:00
2025-07-04	12:00:00
2025-09-01	12:00:00
2025-11-27	12:00:00
2025-11-28	12:15:00
2025-12-24	12:15:00
2025-12-25	16:00:00
2025-12-31	16:00:00
2026-01-19	12:00:00
2026-02-16	12:00:00
2026-04-03	08:15:00
2026-05-25	12:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	12:00:00
2026-11-26	12:00:00

2026-11-27	12:15:00
2026-12-24	12:15:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the E-mini FTSE Emerging Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-07-03	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00

2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The E-mini FTSE Emerging Index Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

EMD

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini S&P MidCap 400 Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini S&P MidCap 400 Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini S&P MidCap 400 Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini S&P MidCap 400 Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the E-mini S&P MidCap 400 Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the E-mini S&P MidCap 400 Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-04-02	08:15:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00
2010-11-26	12:15:00

2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-04-06	08:15:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-20	10:30:00
2014-02-17	10:30:00

2014-05-26	12:00:00
2014-07-03	12:15:00
2014-07-04	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00
2017-07-04	12:00:00

2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00

2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	08:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00
2023-01-16	12:00:00
2023-02-20	12:00:00
2023-04-07	08:15:00
2023-05-29	12:00:00
2023-06-19	12:00:00
2023-07-04	12:00:00
2023-09-04	12:00:00
2023-11-23	12:00:00
2023-11-24	12:15:00
2024-01-15	12:00:00
2024-02-19	12:00:00

2024-05-27	12:00:00
2024-06-19	12:00:00
2024-07-04	12:00:00
2024-09-02	12:00:00
2024-11-28	12:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	12:00:00
2025-02-17	12:00:00
2025-04-17	16:00:00
2025-05-26	12:00:00
2025-06-19	12:00:00
2025-07-03	12:15:00
2025-07-04	12:00:00
2025-09-01	12:00:00
2025-11-27	12:00:00
2025-11-28	12:15:00
2025-12-24	12:15:00
2025-12-25	16:00:00
2025-12-31	16:00:00
2026-01-19	12:00:00
2026-02-16	12:00:00
2026-04-03	08:15:00
2026-05-25	12:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	12:00:00
2026-11-26	12:00:00

2026-11-27	12:15:00
2026-12-24	12:15:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the E-mini S&P MidCap 400 Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-07-03	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00

2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The E-mini S&P MidCap 400 Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

ENY

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini Nikkei YEN Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini Nikkei YEN Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini Nikkei YEN Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini Nikkei YEN Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the E-mini Nikkei YEN Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the E-mini Nikkei YEN Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	11:30:00
2009-02-16	11:30:00
2009-05-25	11:30:00
2009-07-03	11:30:00
2009-09-07	11:30:00
2009-11-26	11:30:00
2009-11-27	13:15:00
2009-12-24	13:15:00
2010-01-18	11:30:00
2010-02-15	11:30:00
2010-04-02	09:15:00
2010-05-31	11:30:00
2010-07-05	11:30:00
2010-09-06	11:30:00
2010-11-25	11:30:00
2010-11-26	13:15:00

2011-01-17	11:30:00
2011-02-21	11:30:00
2011-05-30	11:30:00
2011-07-04	11:30:00
2011-09-05	11:30:00
2011-11-24	11:30:00
2011-11-25	13:15:00
2012-01-16	11:30:00
2012-02-20	11:30:00
2012-04-06	09:15:00
2012-05-28	11:30:00
2012-07-03	13:15:00
2012-07-04	11:30:00
2012-09-03	11:30:00
2012-11-22	11:30:00
2012-11-23	13:15:00
2012-12-24	13:15:00
2013-01-21	11:30:00
2013-02-18	11:30:00
2013-05-27	11:30:00
2013-07-03	13:15:00
2013-07-04	11:30:00
2013-09-02	11:30:00
2013-11-28	11:30:00
2013-11-29	13:15:00
2013-12-24	13:15:00
2014-01-20	11:30:00
2014-02-17	11:30:00

2014-05-26	13:00:00
2014-07-03	13:15:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:15:00
2014-12-24	13:15:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-04-03	09:15:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:15:00
2015-12-24	13:15:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:15:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-03	13:15:00
2017-07-04	13:00:00

2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:15:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-03	13:15:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:15:00
2018-12-24	13:15:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-03	13:15:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:15:00
2019-12-24	13:15:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:15:00

2020-12-24	13:15:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-04-02	09:15:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:15:00
2022-01-17	13:00:00
2022-02-21	13:00:00
2022-05-30	13:00:00
2022-06-20	13:00:00
2022-07-04	13:00:00
2022-09-05	13:00:00
2022-11-24	13:00:00
2022-11-25	13:15:00
2023-01-16	13:00:00
2023-02-20	13:00:00
2023-04-07	09:15:00
2023-05-29	13:00:00
2023-06-19	13:00:00
2023-07-04	13:00:00
2023-09-04	13:00:00
2023-11-23	13:00:00
2023-11-24	13:15:00
2024-01-15	13:00:00
2024-02-19	13:00:00

2024-05-27	13:00:00
2024-06-19	13:00:00
2024-07-04	13:00:00
2024-09-02	13:00:00
2024-11-28	13:00:00
2024-12-24	13:15:00
2024-12-31	17:00:00
2025-01-20	13:00:00
2025-02-17	13:00:00
2025-04-17	17:00:00
2025-05-26	13:00:00
2025-06-19	13:00:00
2025-07-03	13:15:00
2025-07-04	13:00:00
2025-09-01	13:00:00
2025-11-27	13:00:00
2025-11-28	13:15:00
2025-12-24	13:15:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	13:00:00
2026-02-16	13:00:00
2026-04-03	09:15:00
2026-05-25	13:00:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	13:00:00
2026-11-26	13:00:00

2026-11-27	13:15:00
2026-12-24	13:15:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the E-mini Nikkei YEN Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00

2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-07-03	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The E-mini Nikkei YEN Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

ES

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini S&P 500 Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini S&P 500 Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini S&P 500 Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini S&P 500 Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the E-mini S&P 500 Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the E-mini S&P 500 Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	11:30:00
2009-02-16	11:30:00
2009-05-25	11:30:00
2009-07-03	11:30:00
2009-09-07	11:30:00
2009-11-26	11:30:00
2009-11-27	13:15:00
2009-12-24	13:15:00
2010-01-18	11:30:00
2010-02-15	11:30:00
2010-04-02	09:15:00
2010-05-31	11:30:00
2010-07-05	11:30:00
2010-09-06	11:30:00
2010-11-25	11:30:00
2010-11-26	13:15:00

2011-01-17	11:30:00
2011-02-21	11:30:00
2011-05-30	11:30:00
2011-07-04	11:30:00
2011-09-05	11:30:00
2011-11-24	11:30:00
2011-11-25	13:15:00
2012-01-16	11:30:00
2012-02-20	11:30:00
2012-04-06	09:15:00
2012-05-28	11:30:00
2012-07-03	13:15:00
2012-07-04	11:30:00
2012-09-03	11:30:00
2012-11-22	11:30:00
2012-11-23	13:15:00
2012-12-24	13:15:00
2013-01-21	11:30:00
2013-02-18	11:30:00
2013-05-27	11:30:00
2013-07-03	13:15:00
2013-07-04	11:30:00
2013-09-02	11:30:00
2013-11-28	11:30:00
2013-11-29	13:15:00
2013-12-24	13:15:00
2014-01-20	11:30:00
2014-02-17	11:30:00

2014-05-26	13:00:00
2014-07-03	13:15:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:15:00
2014-12-24	13:15:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-04-03	09:15:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:15:00
2015-12-24	13:15:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:15:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-03	13:15:00
2017-07-04	13:00:00

2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:15:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-03	13:15:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:15:00
2018-12-24	13:15:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-03	13:15:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:15:00
2019-12-24	13:15:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:15:00

2020-12-24	13:15:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-04-02	09:15:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:15:00
2022-01-17	13:00:00
2022-02-21	13:00:00
2022-05-30	13:00:00
2022-06-20	13:00:00
2022-07-04	13:00:00
2022-09-05	13:00:00
2022-11-24	13:00:00
2022-11-25	13:15:00
2023-01-16	13:00:00
2023-02-20	13:00:00
2023-04-07	09:15:00
2023-05-29	13:00:00
2023-06-19	13:00:00
2023-07-04	13:00:00
2023-09-04	13:00:00
2023-11-23	13:00:00
2023-11-24	13:15:00
2024-01-15	13:00:00
2024-02-19	13:00:00

2024-05-27	13:00:00
2024-06-19	13:00:00
2024-07-04	13:00:00
2024-09-02	13:00:00
2024-11-28	13:00:00
2024-12-24	13:15:00
2024-12-31	17:00:00
2025-01-20	13:00:00
2025-02-17	13:00:00
2025-04-17	17:00:00
2025-05-26	13:00:00
2025-06-19	13:00:00
2025-07-03	13:15:00
2025-07-04	13:00:00
2025-09-01	13:00:00
2025-11-27	13:00:00
2025-11-28	13:15:00
2025-12-24	13:15:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	13:00:00
2026-02-16	13:00:00
2026-04-03	09:15:00
2026-05-25	13:00:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	13:00:00
2026-11-26	13:00:00

2026-11-27	13:15:00
2026-12-24	13:15:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the E-mini S&P 500 Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00

2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-07-03	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The E-mini S&P 500 Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

ESG

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini S&P500 ESG Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini S&P500 ESG Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini S&P500 ESG Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini S&P500 ESG Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the E-mini S&P500 ESG Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the E-mini S&P500 ESG Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	11:30:00
2009-02-16	11:30:00
2009-05-25	11:30:00
2009-07-03	11:30:00
2009-09-07	11:30:00
2009-11-26	11:30:00
2009-11-27	13:15:00
2009-12-24	13:15:00
2010-01-18	11:30:00
2010-02-15	11:30:00
2010-04-02	09:15:00
2010-05-31	11:30:00
2010-07-05	11:30:00
2010-09-06	11:30:00
2010-11-25	11:30:00
2010-11-26	13:15:00

2011-01-17	11:30:00
2011-02-21	11:30:00
2011-05-30	11:30:00
2011-07-04	11:30:00
2011-09-05	11:30:00
2011-11-24	11:30:00
2011-11-25	13:15:00
2012-01-16	11:30:00
2012-02-20	11:30:00
2012-04-06	09:15:00
2012-05-28	11:30:00
2012-07-03	13:15:00
2012-07-04	11:30:00
2012-09-03	11:30:00
2012-11-22	11:30:00
2012-11-23	13:15:00
2012-12-24	13:15:00
2013-01-21	11:30:00
2013-02-18	11:30:00
2013-05-27	11:30:00
2013-07-03	13:15:00
2013-07-04	11:30:00
2013-09-02	11:30:00
2013-11-28	11:30:00
2013-11-29	13:15:00
2013-12-24	13:15:00
2014-01-20	11:30:00
2014-02-17	11:30:00

2014-05-26	13:00:00
2014-07-03	13:15:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:15:00
2014-12-24	13:15:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-04-03	09:15:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:15:00
2015-12-24	13:15:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:15:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-03	13:15:00
2017-07-04	13:00:00

2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:15:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-03	13:15:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:15:00
2018-12-24	13:15:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-03	13:15:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:15:00
2019-12-24	13:15:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:15:00

2020-12-24	13:15:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-04-02	09:15:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:15:00
2022-01-17	13:00:00
2022-02-21	13:00:00
2022-05-30	13:00:00
2022-06-20	13:00:00
2022-07-04	13:00:00
2022-09-05	13:00:00
2022-11-24	13:00:00
2022-11-25	13:15:00
2023-01-16	13:00:00
2023-02-20	13:00:00
2023-04-07	09:15:00
2023-05-29	13:00:00
2023-06-19	13:00:00
2023-07-04	13:00:00
2023-09-04	13:00:00
2023-11-23	13:00:00
2023-11-24	13:15:00
2024-01-15	13:00:00
2024-02-19	13:00:00

2024-05-27	13:00:00
2024-06-19	13:00:00
2024-07-04	13:00:00
2024-09-02	13:00:00
2024-11-28	13:00:00
2024-12-24	13:15:00
2024-12-31	17:00:00
2025-01-20	13:00:00
2025-02-17	13:00:00
2025-04-17	17:00:00
2025-05-26	13:00:00
2025-06-19	13:00:00
2025-07-03	13:15:00
2025-07-04	13:00:00
2025-09-01	13:00:00
2025-11-27	13:00:00
2025-11-28	13:15:00
2025-12-24	13:15:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	13:00:00
2026-02-16	13:00:00
2026-04-03	09:15:00
2026-05-25	13:00:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	13:00:00
2026-11-26	13:00:00

2026-11-27	13:15:00
2026-12-24	13:15:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the E-mini S&P500 ESG Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00

2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-07-03	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The E-mini S&P500 ESG Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

ESK

Introduction

This page shows the trading hours, holidays, and time zone of the Euro/Swedish Krona Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Euro/Swedish Krona Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Euro/Swedish Krona Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Euro/Swedish Krona Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Euro/Swedish Krona Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Euro/Swedish Krona Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00
2024-01-15	16:00:00

2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00

2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Euro/Swedish Krona Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00

2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Euro/Swedish Krona Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

ETH

Introduction

This page shows the trading hours, holidays, and time zone of the Ether Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Ether Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Ether Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Ether Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Ether Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Ether Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2021-02-15	12:00:00
2021-04-02	08:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:45:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:45:00
2023-01-16	16:00:00

2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:45:00
2024-01-15	16:00:00
2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:45:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00

2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00
2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Ether Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2018-01-15	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00

2020-05-25	17:00:00
2020-07-03	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00

2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Ether Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

FT1

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini FTSE 100 GBP Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini FTSE 100 GBP Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini FTSE 100 GBP Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini FTSE 100 GBP Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the E-mini FTSE 100 GBP Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the E-mini FTSE 100 GBP Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-04-02	08:15:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00
2010-11-26	12:15:00

2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-04-06	08:15:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-20	10:30:00
2014-02-17	10:30:00

2014-05-26	12:00:00
2014-07-03	12:15:00
2014-07-04	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00
2017-07-04	12:00:00

2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00

2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	08:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00
2023-01-16	12:00:00
2023-02-20	12:00:00
2023-04-07	08:15:00
2023-05-29	12:00:00
2023-06-19	12:00:00
2023-07-04	12:00:00
2023-09-04	12:00:00
2023-11-23	12:00:00
2023-11-24	12:15:00
2024-01-15	12:00:00
2024-02-19	12:00:00

2024-05-27	12:00:00
2024-06-19	12:00:00
2024-07-04	12:00:00
2024-09-02	12:00:00
2024-11-28	12:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	12:00:00
2025-02-17	12:00:00
2025-04-17	16:00:00
2025-05-26	12:00:00
2025-06-19	12:00:00
2025-07-03	12:15:00
2025-07-04	12:00:00
2025-09-01	12:00:00
2025-11-27	12:00:00
2025-11-28	12:15:00
2025-12-24	12:15:00
2025-12-25	16:00:00
2025-12-31	16:00:00
2026-01-19	12:00:00
2026-02-16	12:00:00
2026-04-03	08:15:00
2026-05-25	12:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	12:00:00
2026-11-26	12:00:00

2026-11-27	12:15:00
2026-12-24	12:15:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the E-mini FTSE 100 GBP Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-07-03	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00

2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The E-mini FTSE 100 GBP Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

FT5

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini FTSE China Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini FTSE China Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini FTSE China Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini FTSE China Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the E-mini FTSE China Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the E-mini FTSE China Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-04-02	08:15:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00
2010-11-26	12:15:00

2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-04-06	08:15:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-20	10:30:00
2014-02-17	10:30:00

2014-05-26	12:00:00
2014-07-03	12:15:00
2014-07-04	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00
2017-07-04	12:00:00

2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00

2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	08:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00
2023-01-16	12:00:00
2023-02-20	12:00:00
2023-04-07	08:15:00
2023-05-29	12:00:00
2023-06-19	12:00:00
2023-07-04	12:00:00
2023-09-04	12:00:00
2023-11-23	12:00:00
2023-11-24	12:15:00
2024-01-15	12:00:00
2024-02-19	12:00:00

2024-05-27	12:00:00
2024-06-19	12:00:00
2024-07-04	12:00:00
2024-09-02	12:00:00
2024-11-28	12:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	12:00:00
2025-02-17	12:00:00
2025-04-17	16:00:00
2025-05-26	12:00:00
2025-06-19	12:00:00
2025-07-03	12:15:00
2025-07-04	12:00:00
2025-09-01	12:00:00
2025-11-27	12:00:00
2025-11-28	12:15:00
2025-12-24	12:15:00
2025-12-25	16:00:00
2025-12-31	16:00:00
2026-01-19	12:00:00
2026-02-16	12:00:00
2026-04-03	08:15:00
2026-05-25	12:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	12:00:00
2026-11-26	12:00:00

2026-11-27	12:15:00
2026-12-24	12:15:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the E-mini FTSE China Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-07-03	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00

2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The E-mini FTSE China Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

FTU

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini FTSE 100 USD Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini FTSE 100 USD Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini FTSE 100 USD Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini FTSE 100 USD Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the E-mini FTSE 100 USD Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the E-mini FTSE 100 USD Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-04-02	08:15:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00
2010-11-26	12:15:00

2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-04-06	08:15:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-20	10:30:00
2014-02-17	10:30:00

2014-05-26	12:00:00
2014-07-03	12:15:00
2014-07-04	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00
2017-07-04	12:00:00

2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00

2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	08:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00
2023-01-16	12:00:00
2023-02-20	12:00:00
2023-04-07	08:15:00
2023-05-29	12:00:00
2023-06-19	12:00:00
2023-07-04	12:00:00
2023-09-04	12:00:00
2023-11-23	12:00:00
2023-11-24	12:15:00
2024-01-15	12:00:00
2024-02-19	12:00:00

2024-05-27	12:00:00
2024-06-19	12:00:00
2024-07-04	12:00:00
2024-09-02	12:00:00
2024-11-28	12:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	12:00:00
2025-02-17	12:00:00
2025-04-17	16:00:00
2025-05-26	12:00:00
2025-06-19	12:00:00
2025-07-03	12:15:00
2025-07-04	12:00:00
2025-09-01	12:00:00
2025-11-27	12:00:00
2025-11-28	12:15:00
2025-12-24	12:15:00
2025-12-25	16:00:00
2025-12-31	16:00:00
2026-01-19	12:00:00
2026-02-16	12:00:00
2026-04-03	08:15:00
2026-05-25	12:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	12:00:00
2026-11-26	12:00:00

2026-11-27	12:15:00
2026-12-24	12:15:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the E-mini FTSE 100 USD Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-07-03	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00

2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The E-mini FTSE 100 USD Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

GD

Introduction

This page shows the trading hours, holidays, and time zone of the S&P-GSCI Commodity Index Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the S&P-GSCI Commodity Index Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the S&P-GSCI Commodity Index Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the S&P-GSCI Commodity Index Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the S&P-GSCI Commodity Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)				
2023-04-07	2026-04-03	2026-06-19	2026-07-03	2026-12-25
2027-01-01				

Early Closes

The following table shows the early closes for the S&P-GSCI Commodity Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00
2010-11-26	12:15:00

2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-07-05	15:15:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2013-12-26	15:15:00
2014-01-02	15:15:00

2014-01-20	10:30:00
2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00

2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	8:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00
2025-12-31	16:00:00
2026-01-18	16:00:00
2026-02-15	16:00:00
2026-04-02	16:00:00
2026-05-24	16:00:00
2026-06-18	16:00:00
2026-07-02	16:00:00
2026-09-06	16:00:00
2026-11-25	16:00:00
2026-11-27	12:15:00

2026-12-24	12:15:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the S&P-GSCI Commodity Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The S&P-GSCI Commodity Index Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

GDK

Introduction

This page shows the trading hours, holidays, and time zone of the GDK contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the GDK contract in the CME Future market:

Weekday	Time (America/Chicago)
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the GDK contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the GDK contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the GDK contract in the CME Future market:

Date (*yyyy-mm-dd*)				
2014-07-04	2015-04-03	2015-07-03	2017-11-23	2018-11-22
2019-11-28	2020-07-03	2020-11-26	2021-04-02	2021-11-25
2022-11-24	2023-04-07	2023-11-23	2024-11-28	2025-07-04
2025-11-27	2025-11-28	2026-04-03	2026-06-19	2026-07-03
2026-11-26	2026-11-27	2026-12-25	2027-01-01	

Early Closes

The following table shows the early closes for the GDK contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-11-25	16:00:00
2010-12-31	12:15:00
2011-04-21	13:55:00
2011-11-23	16:00:00
2012-01-15	17:00:00
2012-02-19	17:00:00
2012-05-27	17:00:00
2012-07-03	16:00:00
2012-09-02	17:00:00
2012-11-21	16:00:00
2012-12-24	12:00:00
2013-01-20	17:00:00

2013-02-17	17:00:00
2013-03-28	13:55:00
2013-05-26	17:00:00
2013-07-03	12:00:00
2013-09-01	17:00:00
2013-11-27	17:00:00
2013-12-24	12:00:00
2014-02-16	16:00:00
2014-04-17	13:55:00
2014-05-25	16:00:00
2014-07-03	12:00:00
2014-12-24	12:00:00
2015-04-02	13:55:00
2015-07-02	12:00:00
2015-12-24	12:00:00
2015-12-31	13:55:00
2016-03-24	13:55:00
2016-12-23	12:00:00
2017-01-15	16:00:00
2017-02-19	16:00:00
2017-04-13	13:55:00
2017-05-28	16:00:00
2017-07-03	12:00:00
2017-09-03	16:00:00
2017-12-22	12:00:00
2018-01-14	16:00:00
2018-02-18	16:00:00
2018-03-29	13:55:00

2018-05-27	16:00:00
2018-07-03	12:00:00
2018-09-02	16:00:00
2018-12-24	12:00:00
2018-12-31	13:55:00
2019-01-20	16:00:00
2019-02-17	16:00:00
2019-04-18	13:55:00
2019-05-26	16:00:00
2019-07-03	12:00:00
2019-09-01	16:00:00
2019-11-27	16:00:00
2019-12-24	12:00:00
2019-12-31	13:55:00
2020-01-19	16:00:00
2020-02-16	16:00:00
2020-04-09	13:55:00
2020-05-24	16:00:00
2020-07-02	12:00:00
2020-09-06	16:00:00
2020-11-25	16:45:00
2020-12-24	12:00:00
2020-12-31	13:55:00
2021-01-17	16:00:00
2021-02-14	16:00:00
2021-04-01	13:55:00
2021-05-30	16:00:00
2021-07-04	16:00:00

2021-09-05	16:00:00
2022-01-16	16:00:00
2022-02-20	16:00:00
2022-04-14	13:55:00
2022-05-29	16:00:00
2022-06-19	16:00:00
2022-07-03	16:00:00
2022-09-04	16:00:00
2023-01-15	16:00:00
2023-02-19	16:00:00
2023-04-06	13:55:00
2023-05-28	16:00:00
2023-06-18	16:00:00
2023-09-03	16:00:00
2023-11-22	16:00:00
2024-01-01	17:00:00
2024-03-28	13:55:00
2024-11-27	16:00:00
2024-12-24	12:00:00
2024-12-31	16:00:00
2025-01-19	16:00:00
2025-02-16	16:00:00
2025-04-17	13:55:00
2025-05-25	16:00:00
2025-07-03	12:00:00
2025-08-31	16:00:00
2025-11-26	16:00:00
2025-12-24	12:00:00

2025-12-31	16:00:00
2026-01-02	13:55:00
2026-01-18	16:00:00
2026-02-15	16:00:00
2026-04-02	13:55:00
2026-05-24	16:00:00
2026-06-18	13:55:00
2026-07-02	13:55:00
2026-09-06	16:00:00
2026-11-25	16:00:00
2026-12-24	12:00:00
2026-12-31	13:55:00

Late Opens

The following table shows the late opens for the GDK contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2011-12-27	09:05:00
2012-01-03	09:05:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00
2012-12-26	09:05:00
2013-01-02	09:05:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00

2013-09-02	17:00:00
2013-12-26	09:05:00
2014-01-02	09:05:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-12-26	09:05:00
2015-01-02	09:05:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-03-27	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-29	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-05	08:30:00
2017-09-04	17:00:00
2017-11-26	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00

2018-05-28	17:00:00
2018-09-03	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-04-18	13:55:00
2019-05-27	17:00:00
2019-09-02	17:00:00
2019-12-01	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-29	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00

2023-09-04	17:00:00
2023-12-25	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00

Time Zone

The GDK contract in the CME Future market trades in the `America/Chicago` time zone.

CME

GE

Introduction

This page shows the trading hours, holidays, and time zone of the Eurodollar Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Eurodollar Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Eurodollar Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Eurodollar Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04

2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-04-10
2009-12-25	2010-01-01	2010-12-24	2011-01-01	2011-04-22
2011-12-26	2012-01-02	2012-10-29	2012-10-30	2012-12-25
2013-01-01	2013-03-29	2013-12-25	2014-01-01	2014-04-18
2014-12-25	2015-01-01	2015-12-25	2016-01-01	2016-03-25
2016-12-25	2016-12-26	2017-01-02	2017-04-14	2017-12-25
2018-01-01	2018-03-30	2018-12-05	2018-12-25	2019-01-01
2019-04-19	2019-12-25	2020-01-01	2020-04-10	2020-12-25
2021-01-01	2021-12-24	2022-01-01	2022-04-15	2022-12-26
2023-01-02	2023-12-24	2023-12-31	2024-03-29	2025-01-20
2025-04-18				

Early Closes

The following table shows the early closes for the Eurodollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/Chicago)**
2008-12-31	15:15:00
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00

2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00

2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00

2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00

2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00

2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	08:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

The following table shows the late opens for the Eurodollar Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/Chicago)
2009-01-02	05:00:00
2011-12-27	05:00:00
2012-01-03	05:00:00
2012-12-26	05:00:00
2013-01-02	05:00:00
2013-12-26	05:00:00
2014-01-02	05:00:00

Time Zone

The Eurodollar Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

GF

Introduction

This page shows the trading hours, holidays, and time zone of the Feeder Cattle Futures contract in the CME Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Feeder Cattle Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 13:05:00
Tuesday	08:30:00 to 13:05:00
Wednesday	08:30:00 to 13:05:00
Thursday	08:30:00 to 13:05:00
Friday	08:30:00 to 13:05:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Feeder Cattle Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)				
2014-07-04	2015-04-03	2015-07-03	2017-01-16	2017-02-20
2017-05-29	2017-07-04	2017-09-04	2017-11-23	2018-01-15
2018-02-19	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2019-01-21	2019-02-18	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2020-01-20	2020-02-17	2020-05-25	2020-07-03
2020-09-07	2020-11-26	2021-01-18	2021-02-15	2021-04-02
2021-05-31	2021-07-05	2021-09-06	2021-11-25	2022-01-17
2022-02-21	2022-05-30	2022-06-20	2022-07-04	2022-09-05
2022-11-24	2023-01-16	2023-02-20	2023-04-07	2023-05-29
2023-06-19	2023-07-04	2023-09-04	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-02-17
2025-05-26	2025-06-19	2025-07-04	2025-09-01	2025-11-27
2025-12-25	2026-01-01	2026-01-19	2026-02-16	2026-04-03
2026-05-25	2026-06-19	2026-07-03	2026-09-07	2026-11-26
2026-12-25	2027-01-01			

Early Closes

The following table shows the early closes for the Feeder Cattle Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-12-31	12:15:00
2011-11-25	12:15:00
2012-12-24	12:15:00
2013-03-28	13:55:00
2013-07-03	12:15:00
2013-11-27	13:05:00
2013-11-29	12:15:00
2013-12-24	12:15:00

2014-01-19	16:00:00
2014-02-16	16:00:00
2014-04-17	13:55:00
2014-04-20	16:00:00
2014-05-25	16:00:00
2014-07-03	12:15:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-04-02	13:55:00
2015-07-02	12:15:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2015-12-31	13:55:00
2016-11-25	12:15:00
2016-12-23	12:05:00
2017-07-03	12:15:00
2017-11-24	12:15:00
2017-12-22	12:15:00
2018-07-03	12:15:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-07-03	12:15:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-07-02	12:15:00
2020-11-27	12:05:00
2020-12-24	12:05:00
2021-11-26	12:05:00

2022-11-25	12:05:00
2023-11-24	12:05:00
2024-03-28	16:00:00
2024-11-27	16:00:00
2025-11-28	12:05:00
2025-12-24	12:15:00
2026-11-27	12:05:00
2026-12-24	12:15:00

Late Opens

The following table shows the late opens for the Feeder Cattle Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2011-11-24	17:00:00
2011-12-27	09:05:00
2012-01-03	09:05:00
2012-01-17	09:05:00
2012-02-21	09:05:00
2012-05-29	09:05:00
2012-07-05	09:05:00
2012-09-04	09:05:00
2012-11-22	17:00:00
2012-12-26	09:05:00
2013-01-02	09:05:00
2013-01-22	09:05:00
2013-02-19	09:05:00
2013-05-28	09:05:00
2013-07-05	09:05:00
2013-09-03	09:05:00
2013-11-29	09:05:00

2013-12-26	09:05:00
2014-01-02	09:05:00
2014-01-21	09:05:00
2014-02-18	09:05:00
2014-04-21	09:05:00
2014-05-27	09:05:00
2014-07-07	09:05:00
2014-09-02	09:05:00
2014-11-28	08:00:00
2014-12-26	08:00:00
2015-01-02	08:00:00
2015-01-20	09:05:00
2015-02-17	09:05:00
2015-04-06	09:05:00
2015-05-26	09:05:00
2015-07-06	09:05:00
2015-09-08	09:05:00
2015-11-27	08:00:00
2015-12-28	09:05:00
2016-01-04	09:05:00
2016-01-19	09:05:00
2016-02-16	09:05:00
2016-03-28	08:30:00
2016-05-31	08:30:00
2016-07-05	08:30:00
2016-09-06	08:30:00
2016-11-25	08:30:00
2017-04-17	08:30:00

2017-05-30	08:30:00
2017-07-05	08:30:00
2017-09-05	08:30:00
2018-01-16	08:30:00
2018-02-20	08:30:00
2018-04-02	08:30:00
2018-05-29	08:30:00
2018-07-05	08:30:00
2018-09-04	08:30:00
2019-01-22	08:30:00
2019-02-19	08:30:00
2019-04-22	08:30:00
2019-05-28	08:30:00
2019-07-05	08:30:00
2019-09-03	08:30:00
2020-01-21	08:30:00
2020-02-18	08:30:00
2020-04-13	08:30:00
2020-05-26	08:30:00
2020-07-06	08:30:00
2020-09-08	08:30:00
2021-01-19	08:30:00
2021-02-16	08:30:00
2021-04-05	08:30:00
2021-06-01	08:30:00

Time Zone

The Feeder Cattle Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

GNF

Introduction

This page shows the trading hours, holidays, and time zone of the GNF contract in the CME Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GNF contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 13:55:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the GNF contract in the CME Future market:

Date (*yyyy-mm-dd*)				
2014-07-04	2015-04-03	2015-07-03	2017-11-23	2018-11-22
2019-11-28	2020-07-03	2020-11-26	2021-04-02	2021-11-25
2022-11-24	2023-04-07	2023-11-23	2024-11-28	2025-07-04
2025-11-27	2025-11-28	2026-04-03	2026-06-19	2026-07-03
2026-11-26	2026-11-27	2026-12-25	2027-01-01	

Early Closes

The following table shows the early closes for the GNF contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-11-25	16:00:00
2010-12-31	12:15:00
2011-04-21	13:55:00
2011-11-23	16:00:00
2012-01-15	17:00:00
2012-02-19	17:00:00
2012-05-27	17:00:00
2012-07-03	16:00:00
2012-09-02	17:00:00
2012-11-21	16:00:00
2012-12-24	12:00:00
2013-01-20	17:00:00
2013-02-17	17:00:00
2013-03-28	13:55:00
2013-05-26	17:00:00
2013-07-03	12:00:00
2013-09-01	17:00:00
2013-11-27	17:00:00
2013-12-24	12:00:00
2014-02-16	16:00:00
2014-04-17	13:55:00
2014-05-25	16:00:00
2014-07-03	12:00:00
2014-12-24	12:00:00
2015-04-02	13:55:00
2015-07-02	12:00:00
2015-12-24	12:00:00

2015-12-31	13:55:00
2016-03-24	13:55:00
2016-12-23	12:00:00
2017-01-15	16:00:00
2017-02-19	16:00:00
2017-04-13	13:55:00
2017-05-28	16:00:00
2017-07-03	12:00:00
2017-09-03	16:00:00
2017-12-22	12:00:00
2018-01-14	16:00:00
2018-02-18	16:00:00
2018-03-29	13:55:00
2018-05-27	16:00:00
2018-07-03	12:00:00
2018-09-02	16:00:00
2018-12-24	12:00:00
2018-12-31	13:55:00
2019-01-20	16:00:00
2019-02-17	16:00:00
2019-04-18	13:55:00
2019-05-26	16:00:00
2019-07-03	12:00:00
2019-09-01	16:00:00
2019-11-27	16:00:00
2019-12-24	12:00:00
2019-12-31	13:55:00
2020-01-19	16:00:00

2020-02-16	16:00:00
2020-04-09	13:55:00
2020-05-24	16:00:00
2020-07-02	12:00:00
2020-09-06	16:00:00
2020-11-25	16:45:00
2020-12-24	12:00:00
2020-12-31	13:55:00
2021-01-17	16:00:00
2021-02-14	16:00:00
2021-04-01	13:55:00
2021-05-30	16:00:00
2021-07-04	16:00:00
2021-09-05	16:00:00
2022-01-16	16:00:00
2022-02-20	16:00:00
2022-04-14	13:55:00
2022-05-29	16:00:00
2022-06-19	16:00:00
2022-07-03	16:00:00
2022-09-04	16:00:00
2023-01-15	16:00:00
2023-02-19	16:00:00
2023-04-06	13:55:00
2023-05-28	16:00:00
2023-06-18	16:00:00
2023-09-03	16:00:00
2023-11-22	16:00:00

2024-01-01	17:00:00
2024-03-28	13:55:00
2024-11-27	16:00:00
2024-12-24	12:00:00
2024-12-31	16:00:00
2025-01-19	16:00:00
2025-02-16	16:00:00
2025-04-17	13:55:00
2025-05-25	16:00:00
2025-07-03	12:00:00
2025-08-31	16:00:00
2025-11-26	16:00:00
2025-12-24	12:00:00
2025-12-31	16:00:00
2026-01-02	13:55:00
2026-01-18	16:00:00
2026-02-15	16:00:00
2026-04-02	13:55:00
2026-05-24	16:00:00
2026-06-18	13:55:00
2026-07-02	13:55:00
2026-09-06	16:00:00
2026-11-25	16:00:00
2026-12-24	12:00:00
2026-12-31	13:55:00

Late Opens

The following table shows the late opens for the GNF contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**

2011-12-27	09:05:00
2012-01-03	09:05:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00
2012-12-26	09:05:00
2013-01-02	09:05:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-12-26	09:05:00
2014-01-02	09:05:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-12-26	09:05:00
2015-01-02	09:05:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00

2016-03-27	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-29	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-05	08:30:00
2017-09-04	17:00:00
2017-11-26	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-09-03	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-04-18	13:55:00
2019-05-27	17:00:00
2019-09-02	17:00:00
2019-12-01	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-29	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00

2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-09-04	17:00:00
2023-12-25	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00

2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00

Time Zone

The GNF contract in the CME Future market trades in the `America/Chicago` time zone.

CME

HE

Introduction

This page shows the trading hours, holidays, and time zone of the Lean Hog Futures contract in the CME Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Lean Hog Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 13:05:00
Tuesday	08:30:00 to 13:05:00
Wednesday	08:30:00 to 13:05:00
Thursday	08:30:00 to 13:05:00
Friday	08:30:00 to 13:05:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Lean Hog Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)				
2014-07-04	2015-04-03	2015-07-03	2017-01-16	2017-02-20
2017-05-29	2017-07-04	2017-09-04	2017-11-23	2018-01-15
2018-02-19	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2019-01-21	2019-02-18	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2020-01-20	2020-02-17	2020-05-25	2020-07-03
2020-09-07	2020-11-26	2021-01-18	2021-02-15	2021-04-02
2021-05-31	2021-07-05	2021-09-06	2021-11-25	2022-01-17
2022-02-21	2022-05-30	2022-06-20	2022-07-04	2022-09-05
2022-11-24	2023-01-16	2023-02-20	2023-04-07	2023-05-29
2023-06-19	2023-07-04	2023-09-04	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-02-17
2025-05-26	2025-06-19	2025-07-04	2025-09-01	2025-11-27
2025-12-25	2026-01-01	2026-01-19	2026-02-16	2026-04-03
2026-05-25	2026-06-19	2026-07-03	2026-09-07	2026-11-26
2026-12-25	2027-01-01			

Early Closes

The following table shows the early closes for the Lean Hog Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-12-31	12:15:00
2011-11-25	12:15:00
2012-12-24	12:15:00
2013-03-28	13:55:00
2013-07-03	12:15:00
2013-11-27	13:05:00
2013-11-29	12:15:00
2013-12-24	12:15:00

2014-01-19	16:00:00
2014-02-16	16:00:00
2014-04-17	13:55:00
2014-04-20	16:00:00
2014-05-25	16:00:00
2014-07-03	12:15:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-04-02	13:55:00
2015-07-02	12:15:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2015-12-31	13:55:00
2016-11-25	12:15:00
2016-12-23	12:05:00
2017-07-03	12:15:00
2017-11-24	12:15:00
2017-12-22	12:15:00
2018-07-03	12:15:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-07-03	12:15:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-07-02	12:15:00
2020-11-27	12:05:00
2020-12-24	12:05:00
2021-11-26	12:05:00

2022-11-25	12:05:00
2023-11-24	12:05:00
2024-03-28	16:00:00
2024-11-27	16:00:00
2025-11-28	12:05:00
2025-12-24	12:15:00
2026-11-27	12:05:00
2026-12-24	12:15:00

Late Opens

The following table shows the late opens for the Lean Hog Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2011-11-24	17:00:00
2011-12-27	09:05:00
2012-01-03	09:05:00
2012-01-17	09:05:00
2012-02-21	09:05:00
2012-05-29	09:05:00
2012-07-05	09:05:00
2012-09-04	09:05:00
2012-11-22	17:00:00
2012-12-26	09:05:00
2013-01-02	09:05:00
2013-01-22	09:05:00
2013-02-19	09:05:00
2013-05-28	09:05:00
2013-07-05	09:05:00
2013-09-03	09:05:00
2013-11-29	09:05:00

2013-12-26	09:05:00
2014-01-02	09:05:00
2014-01-21	09:05:00
2014-02-18	09:05:00
2014-04-21	09:05:00
2014-05-27	09:05:00
2014-07-07	09:05:00
2014-09-02	09:05:00
2014-11-28	08:00:00
2014-12-26	08:00:00
2015-01-02	08:00:00
2015-01-20	09:05:00
2015-02-17	09:05:00
2015-04-06	09:05:00
2015-05-26	09:05:00
2015-07-06	09:05:00
2015-09-08	09:05:00
2015-11-27	08:00:00
2015-12-28	09:05:00
2016-01-04	09:05:00
2016-01-19	09:05:00
2016-02-16	09:05:00
2016-03-28	08:30:00
2016-05-31	08:30:00
2016-07-05	08:30:00
2016-09-06	08:30:00
2016-11-25	08:30:00
2017-04-17	08:30:00

2017-05-30	08:30:00
2017-07-05	08:30:00
2017-09-05	08:30:00
2018-01-16	08:30:00
2018-02-20	08:30:00
2018-04-02	08:30:00
2018-05-29	08:30:00
2018-07-05	08:30:00
2018-09-04	08:30:00
2019-01-22	08:30:00
2019-02-19	08:30:00
2019-04-22	08:30:00
2019-05-28	08:30:00
2019-07-05	08:30:00
2019-09-03	08:30:00
2020-01-21	08:30:00
2020-02-18	08:30:00
2020-04-13	08:30:00
2020-05-26	08:30:00
2020-07-06	08:30:00
2020-09-08	08:30:00
2021-01-19	08:30:00
2021-02-16	08:30:00
2021-04-05	08:30:00
2021-06-01	08:30:00

Time Zone

The Lean Hog Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

IBV

Introduction

This page shows the trading hours, holidays, and time zone of the USD-Denominated Ibovespa Index Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the USD-Denominated Ibovespa Index Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the USD-Denominated Ibovespa Index Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the USD-Denominated Ibovespa Index Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the USD-Denominated Ibovespa Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the USD-Denominated Ibovespa Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-04-02	08:15:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00

2010-11-26	12:15:00
2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-04-06	08:15:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-20	10:30:00

2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:15:00
2014-07-04	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00

2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	08:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00
2023-01-16	12:00:00
2023-02-20	12:00:00
2023-04-07	08:15:00
2023-05-29	12:00:00
2023-06-19	12:00:00
2023-07-04	12:00:00
2023-09-04	12:00:00
2023-11-23	12:00:00
2023-11-24	12:15:00
2024-01-15	12:00:00

2024-02-19	12:00:00
2024-05-27	12:00:00
2024-06-19	12:00:00
2024-07-04	12:00:00
2024-09-02	12:00:00
2024-11-28	12:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	12:00:00
2025-02-17	12:00:00
2025-04-17	16:00:00
2025-05-26	12:00:00
2025-06-19	12:00:00
2025-07-03	12:15:00
2025-07-04	12:00:00
2025-09-01	12:00:00
2025-11-27	12:00:00
2025-11-28	12:15:00
2025-12-24	12:15:00
2025-12-25	16:00:00
2025-12-31	16:00:00
2026-01-19	12:00:00
2026-02-16	12:00:00
2026-04-03	08:15:00
2026-05-25	12:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	12:00:00

2026-11-26	12:00:00
2026-11-27	12:15:00
2026-12-24	12:15:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the USD-Denominated Ibovespa Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00

2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00

2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00

2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-07-03	17:00:00
2025-09-01	17:00:00

2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The USD-Denominated Ibovespa Index Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

J7

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini Japanese Yen Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini Japanese Yen Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini Japanese Yen Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini Japanese Yen Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the E-mini Japanese Yen Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the E-mini Japanese Yen Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00
2024-01-15	16:00:00

2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00

2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the E-mini Japanese Yen Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00

2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The E-mini Japanese Yen Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

LBR

Introduction

This page shows the trading hours, holidays, and time zone of the Lumber Futures contract in the CME Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Lumber Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	09:00:00 to 15:05:00
Tuesday	09:00:00 to 15:05:00
Wednesday	09:00:00 to 15:05:00
Thursday	09:00:00 to 15:05:00
Friday	09:00:00 to 15:05:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Lumber Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)				
2014-07-04	2015-04-03	2015-07-03	2017-01-16	2017-02-20
2017-05-29	2017-07-04	2017-09-04	2017-11-23	2018-01-15
2018-02-19	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2019-01-21	2019-02-18	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2020-01-20	2020-02-17	2020-05-25	2020-07-03
2020-09-07	2020-11-26	2021-01-18	2021-02-15	2021-04-02
2021-05-31	2021-07-05	2021-09-06	2021-11-25	2022-01-17
2022-02-21	2022-05-30	2022-06-20	2022-07-04	2022-09-05
2022-11-24	2023-01-16	2023-02-20	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2024-01-01	2024-01-15	2024-02-19
2024-05-27	2024-06-19	2024-07-04	2024-09-02	2024-11-28
2024-12-25	2025-01-01	2025-02-17	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Lumber Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-12-31	12:15:00
2013-03-28	13:55:00
2013-07-03	12:00:00
2013-11-27	15:05:00
2013-11-29	12:00:00
2013-12-24	12:00:00
2014-01-19	16:00:00
2014-02-16	16:00:00
2014-04-17	13:55:00

2014-04-20	16:00:00
2014-05-25	16:00:00
2014-07-03	12:00:00
2014-11-28	12:00:00
2014-12-24	12:00:00
2015-04-02	13:55:00
2015-07-02	12:00:00
2015-11-27	12:00:00
2015-12-24	12:00:00
2015-12-31	13:55:00
2016-03-24	13:55:00
2016-11-25	12:00:00
2016-12-23	12:00:00
2017-04-13	13:55:00
2017-07-03	12:00:00
2017-11-24	12:00:00
2017-12-22	12:00:00
2018-03-29	13:55:00
2018-07-03	12:00:00
2018-11-23	12:00:00
2018-12-24	12:00:00
2018-12-31	13:55:00
2019-04-18	13:55:00
2019-07-03	12:05:00
2019-11-29	12:05:00
2019-12-24	12:05:00
2019-12-31	13:55:00
2020-07-02	12:05:00

2020-11-27	12:05:00
2020-12-24	12:05:00
2021-11-26	12:05:00
2022-11-25	12:05:00
2023-11-24	12:05:00
2024-03-28	15:05:00
2024-11-27	15:05:00
2025-11-28	12:05:00
2025-12-24	12:05:00
2026-11-27	12:05:00
2026-12-24	12:15:00

Late Opens

The following table shows the late opens for the Lumber Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2011-11-24	17:00:00
2011-12-27	09:00:00
2012-01-03	09:00:00
2012-01-17	09:00:00
2012-02-21	09:00:00
2012-05-29	09:00:00
2012-07-05	09:00:00
2012-09-04	09:00:00
2012-11-22	17:00:00
2012-12-26	09:00:00
2013-01-02	09:00:00
2013-01-22	09:00:00
2013-02-19	09:00:00
2013-05-28	09:00:00

2013-07-05	09:00:00
2013-09-03	09:00:00
2013-11-29	09:00:00
2013-12-26	09:00:00
2014-01-02	09:00:00
2014-01-21	09:00:00
2014-02-18	09:00:00
2014-04-21	09:00:00
2014-05-27	09:00:00
2014-07-07	09:00:00
2014-09-02	09:00:00
2014-11-28	09:00:00
2014-12-26	09:00:00
2015-01-02	09:00:00
2015-01-20	09:00:00
2015-02-17	09:00:00
2015-04-06	09:00:00
2015-05-26	09:00:00
2015-07-06	09:00:00
2015-09-08	09:00:00
2015-11-27	09:00:00
2015-12-28	09:00:00
2016-01-04	09:00:00
2016-01-19	09:00:00
2016-02-16	09:00:00
2016-03-28	09:00:00
2016-05-31	09:00:00
2016-07-05	09:00:00

2016-09-06	09:00:00
2016-11-25	09:00:00
2017-04-17	09:00:00
2017-05-30	09:00:00
2017-07-05	09:00:00
2017-09-05	09:00:00
2018-01-16	09:00:00
2018-02-20	09:00:00
2018-04-02	09:00:00
2018-05-29	09:00:00
2018-07-05	09:00:00
2018-09-04	09:00:00
2019-01-22	09:00:00
2019-02-19	09:00:00
2019-04-22	09:00:00
2019-05-28	09:00:00
2019-07-05	09:00:00
2019-09-03	09:00:00
2020-01-21	09:00:00
2020-02-18	09:00:00
2020-04-13	09:00:00
2020-05-26	09:00:00
2020-07-06	09:00:00
2020-09-08	09:00:00
2021-01-19	09:00:00
2021-02-16	09:00:00
2021-04-05	09:00:00
2021-06-01	09:00:00

Time Zone

The Lumber Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

LBS

Introduction

This page shows the trading hours, holidays, and time zone of the Random Length Lumber Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Random Length Lumber Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	06:00:00 to 09:00:00
Tuesday	06:00:00 to 09:00:00
Wednesday	06:00:00 to 09:00:00
Thursday	06:00:00 to 09:00:00
Friday	06:00:00 to 09:00:00

Regular Trading Hours

The following table shows the regular trading hours for the Random Length Lumber Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	09:00:00 to 15:05:00
Tuesday	09:00:00 to 15:05:00
Wednesday	09:00:00 to 15:05:00
Thursday	09:00:00 to 15:05:00
Friday	09:00:00 to 15:05:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Random Length Lumber Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)				
2013-01-01	2013-01-21	2013-02-18	2013-03-29	2013-05-27
2013-07-04	2013-09-02	2013-11-28	2014-01-01	2014-01-20
2014-02-17	2014-04-18	2014-04-21	2014-05-26	2014-07-04
2014-09-01	2014-11-27	2015-01-01	2015-01-19	2015-02-16
2015-04-03	2015-05-25	2015-07-03	2015-09-07	2015-11-26
2016-01-01	2016-01-18	2016-02-15	2016-03-25	2016-05-30
2016-07-04	2016-09-05	2016-11-24	2017-01-01	2017-01-02
2017-01-16	2017-02-20	2017-04-14	2017-05-29	2017-07-04
2017-09-04	2017-11-23	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2019-01-01
2019-01-21	2019-02-18	2019-04-19	2019-05-27	2019-07-04
2019-09-02	2019-11-28	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-07-03	2020-09-07	2020-11-26
2021-01-01	2021-01-17	2021-01-18	2021-02-14	2021-02-15
2021-04-02	2021-05-30	2021-05-31	2021-07-04	2021-07-05
2021-09-05	2021-09-06	2021-11-24	2021-11-25	2021-12-26
2022-01-16	2022-01-17	2022-02-20	2022-02-21	2022-04-15
2022-05-29	2022-05-30	2022-06-19	2022-06-20	2022-07-03
2022-07-04	2022-09-04	2022-09-05	2022-11-23	2022-11-24
2022-12-25	2023-01-01	2023-01-16	2023-02-20	2023-04-07
2023-05-29	2023-06-19	2023-07-04	2023-09-04	2023-11-23
2023-12-25	2024-01-01			

Early Closes

The following table shows the early closes for the Random Length Lumber Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2013-07-03	12:00:00
2013-11-29	12:00:00
2013-12-24	12:00:00
2014-07-03	12:00:00
2014-11-28	12:00:00
2014-12-24	12:00:00
2015-07-02	12:00:00
2015-11-27	12:00:00
2015-12-24	12:00:00
2016-11-27	12:00:00
2016-12-24	12:00:00
2017-07-03	12:00:00
2017-11-24	12:00:00
2017-12-24	12:00:00
2018-07-03	12:00:00
2018-11-23	12:00:00
2018-12-24	12:00:00
2019-07-03	12:00:00
2019-11-29	12:05:00
2019-12-24	12:05:00
2020-07-02	12:05:00
2020-11-26	12:05:00
2020-11-27	12:05:00
2020-12-24	12:05:00
2022-11-25	12:05:00

Late Opens

There are no days with late opens.

Time Zone

The Random Length Lumber Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

LE

Introduction

This page shows the trading hours, holidays, and time zone of the Live Cattle Futures contract in the CME Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Live Cattle Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 13:05:00
Tuesday	08:30:00 to 13:05:00
Wednesday	08:30:00 to 13:05:00
Thursday	08:30:00 to 13:05:00
Friday	08:30:00 to 13:05:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Live Cattle Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)				
2014-07-04	2015-04-03	2015-07-03	2017-01-16	2017-02-20
2017-05-29	2017-07-04	2017-09-04	2017-11-23	2018-01-15
2018-02-19	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2019-01-21	2019-02-18	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2020-01-20	2020-02-17	2020-05-25	2020-07-03
2020-09-07	2020-11-26	2021-01-18	2021-02-15	2021-04-02
2021-05-31	2021-07-05	2021-09-06	2021-11-25	2022-01-17
2022-02-21	2022-05-30	2022-06-20	2022-07-04	2022-09-05
2022-11-24	2023-01-16	2023-02-20	2023-04-07	2023-05-29
2023-06-19	2023-07-04	2023-09-04	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-02-17
2025-05-26	2025-06-19	2025-07-04	2025-09-01	2025-11-27
2025-12-25	2026-01-01	2026-01-19	2026-02-16	2026-04-03
2026-05-25	2026-06-19	2026-07-03	2026-09-07	2026-11-26
2026-12-25	2027-01-01			

Early Closes

The following table shows the early closes for the Live Cattle Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-12-31	12:15:00
2011-11-25	12:15:00
2012-12-24	12:15:00
2013-03-28	13:55:00
2013-07-03	12:15:00
2013-11-27	13:05:00
2013-11-29	12:15:00
2013-12-24	12:15:00

2014-01-19	16:00:00
2014-02-16	16:00:00
2014-04-17	13:55:00
2014-04-20	16:00:00
2014-05-25	16:00:00
2014-07-03	12:15:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-04-02	13:55:00
2015-07-02	12:15:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2015-12-31	13:55:00
2016-11-25	12:15:00
2016-12-23	12:05:00
2017-07-03	12:15:00
2017-11-24	12:15:00
2017-12-22	12:15:00
2018-07-03	12:15:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-07-03	12:15:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-07-02	12:15:00
2020-11-27	12:05:00
2020-12-24	12:05:00
2021-11-26	12:05:00

2022-11-25	12:05:00
2023-11-24	12:05:00
2024-03-28	16:00:00
2024-11-27	16:00:00
2025-11-28	12:05:00
2025-12-24	12:15:00
2026-11-27	12:05:00
2026-12-24	12:15:00

Late Opens

The following table shows the late opens for the Live Cattle Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2011-11-24	17:00:00
2011-12-27	09:05:00
2012-01-03	09:05:00
2012-01-17	09:05:00
2012-02-21	09:05:00
2012-05-29	09:05:00
2012-07-05	09:05:00
2012-09-04	09:05:00
2012-11-22	17:00:00
2012-12-26	09:05:00
2013-01-02	09:05:00
2013-01-22	09:05:00
2013-02-19	09:05:00
2013-05-28	09:05:00
2013-07-05	09:05:00
2013-09-03	09:05:00
2013-11-29	09:05:00

2013-12-26	09:05:00
2014-01-02	09:05:00
2014-01-21	09:05:00
2014-02-18	09:05:00
2014-04-21	09:05:00
2014-05-27	09:05:00
2014-07-07	09:05:00
2014-09-02	09:05:00
2014-11-28	08:00:00
2014-12-26	08:00:00
2015-01-02	08:00:00
2015-01-20	09:05:00
2015-02-17	09:05:00
2015-04-06	09:05:00
2015-05-26	09:05:00
2015-07-06	09:05:00
2015-09-08	09:05:00
2015-11-27	08:00:00
2015-12-28	09:05:00
2016-01-04	09:05:00
2016-01-19	09:05:00
2016-02-16	09:05:00
2016-03-28	08:30:00
2016-05-31	08:30:00
2016-07-05	08:30:00
2016-09-06	08:30:00
2016-11-25	08:30:00
2017-04-17	08:30:00

2017-05-30	08:30:00
2017-07-05	08:30:00
2017-09-05	08:30:00
2018-01-16	08:30:00
2018-02-20	08:30:00
2018-04-02	08:30:00
2018-05-29	08:30:00
2018-07-05	08:30:00
2018-09-04	08:30:00
2019-01-22	08:30:00
2019-02-19	08:30:00
2019-04-22	08:30:00
2019-05-28	08:30:00
2019-07-05	08:30:00
2019-09-03	08:30:00
2020-01-21	08:30:00
2020-02-18	08:30:00
2020-04-13	08:30:00
2020-05-26	08:30:00
2020-07-06	08:30:00
2020-09-08	08:30:00
2021-01-19	08:30:00
2021-02-16	08:30:00
2021-04-05	08:30:00
2021-06-01	08:30:00

Time Zone

The Live Cattle Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

M2K

Introduction

This page shows the trading hours, holidays, and time zone of the Micro E-mini Russell 2000 Index Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro E-mini Russell 2000 Index Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro E-mini Russell 2000 Index Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro E-mini Russell 2000 Index Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro E-mini Russell 2000 Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro E-mini Russell 2000 Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	11:30:00
2009-02-16	11:30:00
2009-05-25	11:30:00
2009-07-03	11:30:00
2009-09-07	11:30:00
2009-11-26	11:30:00
2009-11-27	13:15:00
2009-12-24	13:15:00
2010-01-18	11:30:00
2010-02-15	11:30:00
2010-04-02	09:15:00
2010-05-31	11:30:00
2010-07-05	11:30:00
2010-09-06	11:30:00
2010-11-25	11:30:00

2010-11-26	13:15:00
2011-01-17	11:30:00
2011-02-21	11:30:00
2011-05-30	11:30:00
2011-07-04	11:30:00
2011-09-05	11:30:00
2011-11-24	11:30:00
2011-11-25	13:15:00
2012-01-16	11:30:00
2012-02-20	11:30:00
2012-04-06	09:15:00
2012-05-28	11:30:00
2012-07-03	13:15:00
2012-07-04	11:30:00
2012-09-03	11:30:00
2012-11-22	11:30:00
2012-11-23	13:15:00
2012-12-24	13:15:00
2013-01-21	11:30:00
2013-02-18	11:30:00
2013-05-27	11:30:00
2013-07-03	13:15:00
2013-07-04	11:30:00
2013-09-02	11:30:00
2013-11-28	11:30:00
2013-11-29	13:15:00
2013-12-24	13:15:00
2014-01-20	11:30:00

2014-02-17	11:30:00
2014-05-26	13:00:00
2014-07-03	13:15:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:15:00
2014-12-24	13:15:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-04-03	09:15:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:15:00
2015-12-24	13:15:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:15:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-03	13:15:00

2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:15:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-03	13:15:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:15:00
2018-12-24	13:15:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-03	13:15:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:15:00
2019-12-24	13:15:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00
2020-11-26	13:00:00

2020-11-27	13:15:00
2020-12-24	13:15:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-04-02	09:15:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:15:00
2022-01-17	13:00:00
2022-02-21	13:00:00
2022-05-30	13:00:00
2022-06-20	13:00:00
2022-07-04	13:00:00
2022-09-05	13:00:00
2022-11-24	13:00:00
2022-11-25	13:15:00
2023-01-16	13:00:00
2023-02-20	13:00:00
2023-04-07	09:15:00
2023-05-29	13:00:00
2023-06-19	13:00:00
2023-07-04	13:00:00
2023-09-04	13:00:00
2023-11-23	13:00:00
2023-11-24	13:15:00
2024-01-15	13:00:00

2024-02-19	13:00:00
2024-05-27	13:00:00
2024-06-19	13:00:00
2024-07-04	13:00:00
2024-09-02	13:00:00
2024-11-28	13:00:00
2024-12-24	13:15:00
2024-12-31	17:00:00
2025-01-20	13:00:00
2025-02-17	13:00:00
2025-04-17	17:00:00
2025-05-26	13:00:00
2025-06-19	13:00:00
2025-07-03	13:15:00
2025-07-04	13:00:00
2025-09-01	13:00:00
2025-11-27	13:00:00
2025-11-28	13:15:00
2025-12-24	13:15:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	13:00:00
2026-02-16	13:00:00
2026-04-03	09:15:00
2026-05-25	13:00:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	13:00:00

2026-11-26	13:00:00
2026-11-27	13:15:00
2026-12-24	13:15:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Micro E-mini Russell 2000 Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00

2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00

2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00

2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-07-03	18:00:00
2025-09-01	18:00:00

2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Micro E-mini Russell 2000 Index Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

M6A

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Australian Dollar/U.S. Dollar (AUD/USD) Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Australian Dollar/U.S. Dollar (AUD/USD) Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Australian Dollar/U.S. Dollar (AUD/USD) Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Australian Dollar/U.S. Dollar (AUD/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro Australian Dollar/U.S. Dollar (AUD/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro Australian Dollar/U.S. Dollar (AUD/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00

2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00

2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00

2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00

2024-01-15	16:00:00
2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00

2026-11-26	16:00:00
2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Micro Australian Dollar/U.S. Dollar (AUD/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00

2012-07-04	17:00:00
2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00

2017-07-04	17:00:00
2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00

2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00

2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Micro Australian Dollar/U.S. Dollar (AUD/USD) Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

M6B

Introduction

This page shows the trading hours, holidays, and time zone of the Micro British Pound Sterling/U.S. Dollar (GBP/USD) Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro British Pound Sterling/U.S. Dollar (GBP/USD) Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro British Pound Sterling/U.S. Dollar (GBP/USD) Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro British Pound Sterling/U.S. Dollar (GBP/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro British Pound Sterling/U.S. Dollar (GBP/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro British Pound Sterling/U.S. Dollar (GBP/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00

2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00

2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00

2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00

2024-01-15	16:00:00
2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00

2026-11-26	16:00:00
2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Micro British Pound Sterling/U.S. Dollar (GBP/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00

2012-07-04	17:00:00
2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00

2017-07-04	17:00:00
2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00

2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00

2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Micro British Pound Sterling/U.S. Dollar (GBP/USD) Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

M6C

Introduction

This page shows the trading hours, holidays, and time zone of the Micro USD/CAD Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro USD/CAD Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro USD/CAD Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro USD/CAD Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04

2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-04-10
2009-12-25	2010-01-01	2010-12-24	2011-01-01	2011-04-22
2011-12-26	2012-01-02	2012-10-29	2012-10-30	2012-12-25
2013-01-01	2013-03-29	2013-12-25	2014-01-01	2014-04-18
2014-12-25	2015-01-01	2015-12-25	2016-01-01	2016-03-25
2016-12-25	2016-12-26	2017-01-02	2017-04-14	2017-12-25
2018-01-01	2018-03-30	2018-12-05	2018-12-25	2019-01-01
2019-04-19	2019-12-25	2020-01-01	2020-04-10	2020-12-25
2021-01-01	2021-12-24	2022-01-01	2022-04-15	2022-12-26
2023-01-02	2023-12-24	2023-12-31	2024-03-29	2025-01-20
2025-04-18				

Early Closes

The following table shows the early closes for the Micro USD/CAD Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/Chicago)**
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00

2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00

2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00

2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00

2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00

2023-11-24	12:15:00
2024-01-15	16:00:00
2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00

Late Opens

The following table shows the late opens for the Micro USD/CAD Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00

2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00

2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00

2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00

2025-01-01	17:00:00

Time Zone

The Micro USD/CAD Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

M6E

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Euro/U.S. Dollar (EUR/USD) Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Euro/U.S. Dollar (EUR/USD) Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Euro/U.S. Dollar (EUR/USD) Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Euro/U.S. Dollar (EUR/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro Euro/U.S. Dollar (EUR/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro Euro/U.S. Dollar (EUR/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00

2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00

2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00

2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00

2024-01-15	16:00:00
2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00

2026-11-26	16:00:00
2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Micro Euro/U.S. Dollar (EUR/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00

2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00

2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00

2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00

2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Micro Euro/U.S. Dollar (EUR/USD) Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

M6J

Introduction

This page shows the trading hours, holidays, and time zone of the Micro USD/JPY Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro USD/JPY Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro USD/JPY Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro USD/JPY Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04

2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-04-10
2009-12-25	2010-01-01	2010-12-24	2011-01-01	2011-04-22
2011-12-26	2012-01-02	2012-10-29	2012-10-30	2012-12-25
2013-01-01	2013-03-29	2013-12-25	2014-01-01	2014-04-18
2014-12-25	2015-01-01	2015-12-25	2016-01-01	2016-03-25
2016-12-25	2016-12-26	2017-01-02	2017-04-14	2017-12-25
2018-01-01	2018-03-30	2018-12-05	2018-12-25	2019-01-01
2019-04-19	2019-12-25	2020-01-01	2020-04-10	2020-12-25
2021-01-01	2021-12-24	2022-01-01	2022-04-15	2022-12-26
2023-01-02	2023-12-24	2023-12-31	2024-03-29	2025-01-20
2025-04-18				

Early Closes

The following table shows the early closes for the Micro USD/JPY Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/Chicago)**
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00

2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00

2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00

2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00

2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00

2023-11-24	12:15:00
2024-01-15	16:00:00
2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00

Late Opens

The following table shows the late opens for the Micro USD/JPY Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00

2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00

2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00

2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00

2025-01-01	17:00:00

Time Zone

The Micro USD/JPY Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

M6S

Introduction

This page shows the trading hours, holidays, and time zone of the Micro USD/CHF Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro USD/CHF Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro USD/CHF Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro USD/CHF Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04

2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-04-10
2009-12-25	2010-01-01	2010-12-24	2011-01-01	2011-04-22
2011-12-26	2012-01-02	2012-10-29	2012-10-30	2012-12-25
2013-01-01	2013-03-29	2013-12-25	2014-01-01	2014-04-18
2014-12-25	2015-01-01	2015-12-25	2016-01-01	2016-03-25
2016-12-25	2016-12-26	2017-01-02	2017-04-14	2017-12-25
2018-01-01	2018-03-30	2018-12-05	2018-12-25	2019-01-01
2019-04-19	2019-12-25	2020-01-01	2020-04-10	2020-12-25
2021-01-01	2021-12-24	2022-01-01	2022-04-15	2022-12-26
2023-01-02	2023-12-24	2023-12-31	2024-03-29	2025-01-20
2025-04-18				

Early Closes

The following table shows the early closes for the Micro USD/CHF Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/Chicago)**
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00

2010-01-15	15:15:00
2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00

2012-01-13	15:15:00
2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00

2014-02-14	15:15:00
2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00

2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00

2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00

2023-11-24	12:15:00
2024-01-15	16:00:00
2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00

Late Opens

The following table shows the late opens for the Micro USD/CHF Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00

2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00

2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00

2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00

2025-01-01	17:00:00

Time Zone

The Micro USD/CHF Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

MBT

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Bitcoin Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Bitcoin Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Bitcoin Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Bitcoin Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro Bitcoin Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro Bitcoin Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:45:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:45:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00

2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:45:00
2024-01-15	16:00:00
2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:45:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00

2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00
2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Micro Bitcoin Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2018-01-15	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-07-03	17:00:00

2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00

2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Micro Bitcoin Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

MCD

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Canadian Dollar/U.S.Dollar(CAD/USD) Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Canadian Dollar/U.S.Dollar(CAD/USD) Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Canadian Dollar/U.S.Dollar(CAD/USD) Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Canadian Dollar/U.S.Dollar(CAD/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro Canadian Dollar/U.S.Dollar(CAD/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro Canadian Dollar/U.S.Dollar(CAD/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00

2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00

2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00

2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00

2024-01-15	16:00:00
2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00

2026-11-26	16:00:00
2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Micro Canadian Dollar/U.S.Dollar(CAD/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00

2012-07-04	17:00:00
2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00

2017-07-04	17:00:00
2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00

2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00

2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Micro Canadian Dollar/U.S.Dollar(CAD/USD) Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

MES

Introduction

This page shows the trading hours, holidays, and time zone of the Micro E-mini Standard and Poor's 500 Stock Price Index Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro E-mini Standard and Poor's 500 Stock Price Index Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro E-mini Standard and Poor's 500 Stock Price Index Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro E-mini Standard and Poor's 500 Stock Price Index Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro E-mini Standard and Poor's 500 Stock Price Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro E-mini Standard and Poor's 500 Stock Price Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	11:30:00
2009-02-16	11:30:00
2009-05-25	11:30:00
2009-07-03	11:30:00
2009-09-07	11:30:00
2009-11-26	11:30:00
2009-11-27	13:15:00
2009-12-24	13:15:00
2010-01-18	11:30:00
2010-02-15	11:30:00
2010-04-02	09:15:00
2010-05-31	11:30:00
2010-07-05	11:30:00
2010-09-06	11:30:00
2010-11-25	11:30:00

2010-11-26	13:15:00
2011-01-17	11:30:00
2011-02-21	11:30:00
2011-05-30	11:30:00
2011-07-04	11:30:00
2011-09-05	11:30:00
2011-11-24	11:30:00
2011-11-25	13:15:00
2012-01-16	11:30:00
2012-02-20	11:30:00
2012-04-06	09:15:00
2012-05-28	11:30:00
2012-07-03	13:15:00
2012-07-04	11:30:00
2012-09-03	11:30:00
2012-11-22	11:30:00
2012-11-23	13:15:00
2012-12-24	13:15:00
2013-01-21	11:30:00
2013-02-18	11:30:00
2013-05-27	11:30:00
2013-07-03	13:15:00
2013-07-04	11:30:00
2013-09-02	11:30:00
2013-11-28	11:30:00
2013-11-29	13:15:00
2013-12-24	13:15:00
2014-01-20	11:30:00

2014-02-17	11:30:00
2014-05-26	13:00:00
2014-07-03	13:15:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:15:00
2014-12-24	13:15:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-04-03	09:15:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:15:00
2015-12-24	13:15:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:15:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-03	13:15:00

2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:15:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-03	13:15:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:15:00
2018-12-24	13:15:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-03	13:15:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:15:00
2019-12-24	13:15:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00
2020-11-26	13:00:00

2020-11-27	13:15:00
2020-12-24	13:15:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-04-02	09:15:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:15:00
2022-01-17	13:00:00
2022-02-21	13:00:00
2022-05-30	13:00:00
2022-06-20	13:00:00
2022-07-04	13:00:00
2022-09-05	13:00:00
2022-11-24	13:00:00
2022-11-25	13:15:00
2023-01-16	13:00:00
2023-02-20	13:00:00
2023-04-07	09:15:00
2023-05-29	13:00:00
2023-06-19	13:00:00
2023-07-04	13:00:00
2023-09-04	13:00:00
2023-11-23	13:00:00
2023-11-24	13:15:00
2024-01-15	13:00:00

2024-02-19	13:00:00
2024-05-27	13:00:00
2024-06-19	13:00:00
2024-07-04	13:00:00
2024-09-02	13:00:00
2024-11-28	13:00:00
2024-12-24	13:15:00
2024-12-31	17:00:00
2025-01-20	13:00:00
2025-02-17	13:00:00
2025-04-17	17:00:00
2025-05-26	13:00:00
2025-06-19	13:00:00
2025-07-03	13:15:00
2025-07-04	13:00:00
2025-09-01	13:00:00
2025-11-27	13:00:00
2025-11-28	13:15:00
2025-12-24	13:15:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	13:00:00
2026-02-16	13:00:00
2026-04-03	09:15:00
2026-05-25	13:00:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	13:00:00

2026-11-26	13:00:00
2026-11-27	13:15:00
2026-12-24	13:15:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Micro E-mini Standard and Poor's 500 Stock Price Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00

2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-07-03	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Micro E-mini Standard and Poor's 500 Stock Price Index Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

MET

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Ether Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Ether Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Ether Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Ether Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro Ether Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro Ether Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:45:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00

2023-11-24	12:45:00
2024-01-15	16:00:00
2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:45:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00

2026-09-07	16:00:00
2026-11-26	16:00:00
2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Micro Ether Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2018-01-15	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-07-03	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00

2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00

2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Micro Ether Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

MIB

Introduction

This page shows the trading hours, holidays, and time zone of the BTIC on Micro Bitcoin Futures contract in the CME Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the BTIC on Micro Bitcoin Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the BTIC on Micro Bitcoin Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)				
2010-04-02	2012-04-06	2013-01-02	2022-12-25	2023-01-01
2025-01-09	2026-12-25	2027-01-01		

Early Closes

The following table shows the early closes for the BTIC on Micro Bitcoin Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/New York)**
2009-01-19	10:30:00

2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00
2010-11-26	12:15:00
2011-01-17	10:30:00
2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00

2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-07-05	15:15:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2013-12-26	15:15:00
2014-01-02	15:15:00
2014-01-20	10:30:00
2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00

2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00

2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	8:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00

2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-03-28	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-04-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-07-04	13:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-31	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-04-03	11:15:00

2026-05-25	17:00:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the BTIC on Micro Bitcoin Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The BTIC on Micro Bitcoin Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

MIR

Introduction

This page shows the trading hours, holidays, and time zone of the Micro INR/USD Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro INR/USD Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro INR/USD Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro INR/USD Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro INR/USD Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro INR/USD Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00
2024-01-15	16:00:00

2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00

2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Micro INR/USD Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00

2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Micro INR/USD Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

MJY

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Japanese Yen/U.S. Dollar (JPY/USD) Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Japanese Yen/U.S. Dollar (JPY/USD) Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Japanese Yen/U.S. Dollar (JPY/USD) Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Japanese Yen/U.S. Dollar (JPY/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro Japanese Yen/U.S. Dollar (JPY/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro Japanese Yen/U.S. Dollar (JPY/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-02	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:00:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:12:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00

2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00

2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00

2020-12-24	12:15:00
2021-01-17	16:00:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	12:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00
2024-01-15	16:00:00
2024-02-19	16:00:00

2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00
2026-11-27	13:45:00

2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Micro Japanese Yen/U.S. Dollar (JPY/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/Chicago)
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Micro Japanese Yen/U.S. Dollar (JPY/USD) Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

MNH

Introduction

This page shows the trading hours, holidays, and time zone of the Micro USD/CNH Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro USD/CNH Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro USD/CNH Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro USD/CNH Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro USD/CNH Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro USD/CNH Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00
2010-01-18	12:00:00

2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00
2012-01-16	12:00:00

2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00
2014-02-17	12:00:00

2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00

2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00

2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00
2024-01-15	16:00:00

2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00
2026-11-26	16:00:00

2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Micro USD/CNH Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00
2012-07-04	17:00:00
2012-09-03	17:00:00

2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00
2017-07-04	17:00:00
2017-09-04	17:00:00

2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00
2022-05-30	17:00:00
2022-06-20	17:00:00

2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00

2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Micro USD/CNH Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

MNQ

Introduction

This page shows the trading hours, holidays, and time zone of the Micro E-mini Nasdaq-100 Index Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro E-mini Nasdaq-100 Index Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro E-mini Nasdaq-100 Index Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro E-mini Nasdaq-100 Index Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro E-mini Nasdaq-100 Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro E-mini Nasdaq-100 Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	11:30:00
2009-02-16	11:30:00
2009-05-25	11:30:00
2009-07-03	11:30:00
2009-09-07	11:30:00
2009-11-26	11:30:00
2009-11-27	13:15:00
2009-12-24	13:15:00
2010-01-18	11:30:00
2010-02-15	11:30:00
2010-04-02	09:15:00
2010-05-31	11:30:00
2010-07-05	11:30:00
2010-09-06	11:30:00
2010-11-25	11:30:00

2010-11-26	13:15:00
2011-01-17	11:30:00
2011-02-21	11:30:00
2011-05-30	11:30:00
2011-07-04	11:30:00
2011-09-05	11:30:00
2011-11-24	11:30:00
2011-11-25	13:15:00
2012-01-16	11:30:00
2012-02-20	11:30:00
2012-04-06	09:15:00
2012-05-28	11:30:00
2012-07-03	13:15:00
2012-07-04	11:30:00
2012-09-03	11:30:00
2012-11-22	11:30:00
2012-11-23	13:15:00
2012-12-24	13:15:00
2013-01-21	11:30:00
2013-02-18	11:30:00
2013-05-27	11:30:00
2013-07-03	13:15:00
2013-07-04	11:30:00
2013-09-02	11:30:00
2013-11-28	11:30:00
2013-11-29	13:15:00
2013-12-24	13:15:00
2014-01-20	11:30:00

2014-02-17	11:30:00
2014-05-26	13:00:00
2014-07-03	13:15:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:15:00
2014-12-24	13:15:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-04-03	09:15:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:15:00
2015-12-24	13:15:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:15:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-03	13:15:00

2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:15:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-03	13:15:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:15:00
2018-12-24	13:15:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-03	13:15:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:15:00
2019-12-24	13:15:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00
2020-11-26	13:00:00

2020-11-27	13:15:00
2020-12-24	13:15:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-04-02	09:15:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:15:00
2022-01-17	13:00:00
2022-02-21	13:00:00
2022-05-30	13:00:00
2022-06-20	13:00:00
2022-07-04	13:00:00
2022-09-05	13:00:00
2022-11-24	13:00:00
2022-11-25	13:15:00
2023-01-16	13:00:00
2023-02-20	13:00:00
2023-04-07	09:15:00
2023-05-29	13:00:00
2023-06-19	13:00:00
2023-07-04	13:00:00
2023-09-04	13:00:00
2023-11-23	13:00:00
2023-11-24	13:15:00
2024-01-15	13:00:00

2024-02-19	13:00:00
2024-05-27	13:00:00
2024-06-19	13:00:00
2024-07-04	13:00:00
2024-09-02	13:00:00
2024-11-28	13:00:00
2024-12-24	13:15:00
2024-12-31	17:00:00
2025-01-20	13:00:00
2025-02-17	13:00:00
2025-04-17	17:00:00
2025-05-26	13:00:00
2025-06-19	13:00:00
2025-07-03	13:15:00
2025-07-04	13:00:00
2025-09-01	13:00:00
2025-11-27	13:00:00
2025-11-28	13:15:00
2025-12-24	13:15:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	13:00:00
2026-02-16	13:00:00
2026-04-03	09:15:00
2026-05-25	13:00:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	13:00:00

2026-11-26	13:00:00
2026-11-27	13:15:00
2026-12-24	13:15:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Micro E-mini Nasdaq-100 Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00

2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00

2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00

2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-07-03	18:00:00
2025-09-01	18:00:00

2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Micro E-mini Nasdaq-100 Index Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

MRB

Introduction

This page shows the trading hours, holidays, and time zone of the BTIC on Micro Ether Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the BTIC on Micro Ether Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the BTIC on Micro Ether Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the BTIC on Micro Ether Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the BTIC on Micro Ether Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)				
2010-04-02	2012-04-06	2013-01-02	2022-12-25	2023-01-01

Early Closes

The following table shows the early closes for the BTIC on Micro Ether Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	10:30:00
2009-02-16	10:30:00
2009-05-25	10:30:00
2009-07-03	10:30:00
2009-09-07	10:30:00
2009-11-26	10:30:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-18	10:30:00
2010-02-15	10:30:00
2010-05-31	10:30:00
2010-07-05	10:30:00
2010-09-06	10:30:00
2010-11-25	10:30:00
2010-11-26	12:15:00
2011-01-17	10:30:00

2011-02-21	10:30:00
2011-05-30	10:30:00
2011-07-04	10:30:00
2011-09-05	10:30:00
2011-11-24	10:30:00
2011-11-25	12:15:00
2012-01-16	10:30:00
2012-02-20	10:30:00
2012-05-28	10:30:00
2012-07-03	12:15:00
2012-07-04	10:30:00
2012-09-03	10:30:00
2012-11-22	10:30:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-21	10:30:00
2013-02-18	10:30:00
2013-05-27	10:30:00
2013-07-03	12:15:00
2013-07-04	10:30:00
2013-07-05	15:15:00
2013-09-02	10:30:00
2013-11-28	10:30:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2013-12-26	15:15:00
2014-01-02	15:15:00
2014-01-20	10:30:00

2014-02-17	10:30:00
2014-05-26	12:00:00
2014-07-03	12:00:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-19	12:00:00
2015-02-16	12:00:00
2015-04-03	08:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00
2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-03	12:15:00
2017-07-04	12:00:00

2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-03	12:15:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-03	12:15:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:15:00

2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	8:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	12:00:00
2022-02-21	12:00:00
2022-05-30	12:00:00
2022-06-20	12:00:00
2022-07-04	12:00:00
2022-09-05	12:00:00
2022-11-24	12:00:00
2022-11-25	12:15:00

Late Opens

There are no days with late opens.

Time Zone

The BTIC on Micro Ether Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

MSF

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Swiss Franc/U.S. Dollar (CHF/USD) Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Swiss Franc/U.S. Dollar (CHF/USD) Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Swiss Franc/U.S. Dollar (CHF/USD) Futures contract in the CME Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Swiss Franc/U.S. Dollar (CHF/USD) Futures contract in the CME Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro Swiss Franc/U.S. Dollar (CHF/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)		
2026-07-04	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro Swiss Franc/U.S. Dollar (CHF/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2009-01-16	15:15:00
2009-01-19	12:00:00
2009-02-13	15:15:00
2009-02-16	12:00:00
2009-04-09	15:15:00
2009-05-22	15:15:00
2009-05-25	12:00:00
2009-07-03	12:00:00
2009-09-04	15:15:00
2009-09-07	12:00:00
2009-10-09	15:15:00
2009-11-26	12:00:00
2009-11-27	12:15:00
2009-12-24	12:15:00
2010-01-15	15:15:00

2010-01-18	12:00:00
2010-02-12	15:15:00
2010-02-15	12:00:00
2010-04-02	10:15:00
2010-05-28	15:15:00
2010-05-31	12:00:00
2010-07-02	15:15:00
2010-07-05	12:00:00
2010-09-03	15:15:00
2010-09-06	12:00:00
2010-10-08	15:15:00
2010-11-25	12:00:00
2010-11-26	12:15:00
2010-12-31	12:15:00
2011-01-14	15:15:00
2011-01-17	12:00:00
2011-02-18	15:15:00
2011-02-21	12:00:00
2011-05-27	15:15:00
2011-05-30	12:00:00
2011-07-01	15:15:00
2011-07-04	12:00:00
2011-09-02	15:15:00
2011-09-05	12:00:00
2011-10-07	15:15:00
2011-11-24	12:00:00
2011-11-25	12:15:00
2012-01-13	15:15:00

2012-01-16	12:00:00
2012-02-17	15:15:00
2012-02-20	12:00:00
2012-04-06	10:15:00
2012-05-25	15:15:00
2012-05-28	12:00:00
2012-07-04	12:00:00
2012-08-31	15:15:00
2012-09-03	12:00:00
2012-10-05	15:15:00
2012-11-22	12:00:00
2012-11-23	12:15:00
2012-12-24	12:15:00
2013-01-18	15:15:00
2013-01-21	12:00:00
2013-02-15	15:15:00
2013-02-18	12:00:00
2013-05-24	15:15:00
2013-05-27	12:00:00
2013-07-04	12:00:00
2013-08-30	15:15:00
2013-09-02	12:00:00
2013-11-28	12:00:00
2013-11-29	12:15:00
2013-12-24	12:15:00
2014-01-17	15:15:00
2014-01-20	12:00:00
2014-02-14	15:15:00

2014-02-17	12:00:00
2014-05-23	15:15:00
2014-05-26	12:00:00
2014-07-04	12:00:00
2014-08-29	15:15:00
2014-09-01	12:00:00
2014-11-27	12:00:00
2014-11-28	12:15:00
2014-12-24	12:15:00
2015-01-16	15:15:00
2015-01-19	12:00:00
2015-02-13	15:15:00
2015-02-16	12:00:00
2015-04-03	10:15:00
2015-05-22	15:15:00
2015-05-25	12:00:00
2015-07-03	12:00:00
2015-09-07	12:00:00
2015-11-26	12:00:00
2015-11-27	12:15:00
2015-12-24	12:15:00
2016-01-18	12:00:00
2016-02-15	12:00:00
2016-05-30	12:00:00
2016-07-04	12:00:00
2016-09-05	12:00:00
2016-11-24	12:00:00
2016-11-25	12:15:00

2017-01-16	12:00:00
2017-02-20	12:00:00
2017-05-29	12:00:00
2017-07-04	12:00:00
2017-09-04	12:00:00
2017-11-23	12:00:00
2017-11-24	12:15:00
2018-01-15	12:00:00
2018-02-19	12:00:00
2018-05-28	12:00:00
2018-07-04	12:00:00
2018-09-03	12:00:00
2018-11-22	12:00:00
2018-11-23	12:15:00
2018-12-24	12:15:00
2019-01-21	12:00:00
2019-02-18	12:00:00
2019-05-27	12:00:00
2019-07-04	12:00:00
2019-09-02	12:00:00
2019-11-28	12:00:00
2019-11-29	12:15:00
2019-12-24	12:15:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00

2020-11-26	12:00:00
2020-11-27	12:15:00
2020-12-24	12:15:00
2021-01-18	12:00:00
2021-02-15	12:00:00
2021-04-02	10:15:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:15:00
2022-01-17	16:00:00
2022-02-21	16:00:00
2022-05-30	16:00:00
2022-06-20	16:00:00
2022-07-04	16:00:00
2022-09-05	16:00:00
2022-11-24	16:00:00
2022-11-25	12:15:00
2023-01-16	16:00:00
2023-02-20	16:00:00
2023-04-07	10:15:00
2023-05-29	16:00:00
2023-06-19	16:00:00
2023-07-04	16:00:00
2023-09-04	16:00:00
2023-11-23	16:00:00
2023-11-24	12:15:00

2024-01-15	16:00:00
2024-02-19	16:00:00
2024-03-28	16:00:00
2024-05-27	16:00:00
2024-06-19	16:00:00
2024-07-04	16:00:00
2024-09-02	16:00:00
2024-11-28	16:00:00
2024-12-24	12:15:00
2024-12-31	16:00:00
2025-01-20	16:00:00
2025-02-17	16:00:00
2025-04-17	16:00:00
2025-05-26	16:00:00
2025-06-19	16:00:00
2025-07-04	12:00:00
2025-09-01	16:00:00
2025-11-27	16:00:00
2025-11-28	13:45:00
2025-12-24	12:45:00
2025-12-31	16:00:00
2026-01-19	16:00:00
2026-02-16	16:00:00
2026-04-03	10:15:00
2026-05-25	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-07	16:00:00

2026-11-26	16:00:00
2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the Micro Swiss Franc/U.S. Dollar (CHF/USD) Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2009-01-19	17:00:00
2009-02-16	17:00:00
2009-05-25	17:00:00
2009-09-07	17:00:00
2009-11-26	17:00:00
2010-01-18	17:00:00
2010-02-15	17:00:00
2010-05-31	17:00:00
2010-07-05	17:00:00
2010-09-06	17:00:00
2010-11-25	17:00:00
2011-01-17	17:00:00
2011-02-21	17:00:00
2011-05-30	17:00:00
2011-07-04	17:00:00
2011-09-05	17:00:00
2011-11-24	17:00:00
2012-01-16	17:00:00
2012-02-20	17:00:00
2012-05-28	17:00:00

2012-07-04	17:00:00
2012-09-03	17:00:00
2012-11-22	17:00:00
2013-01-21	17:00:00
2013-02-18	17:00:00
2013-05-27	17:00:00
2013-07-04	17:00:00
2013-09-02	17:00:00
2013-11-28	17:00:00
2014-01-20	17:00:00
2014-02-17	17:00:00
2014-05-26	17:00:00
2014-09-01	17:00:00
2014-11-27	17:00:00
2015-01-19	17:00:00
2015-02-16	17:00:00
2015-05-25	17:00:00
2015-09-07	17:00:00
2015-11-26	17:00:00
2016-01-18	17:00:00
2016-02-15	17:00:00
2016-05-30	17:00:00
2016-07-04	17:00:00
2016-09-05	17:00:00
2016-11-24	17:00:00
2017-01-16	17:00:00
2017-02-20	17:00:00
2017-05-29	17:00:00

2017-07-04	17:00:00
2017-09-04	17:00:00
2017-11-23	17:00:00
2018-01-15	17:00:00
2018-02-19	17:00:00
2018-05-28	17:00:00
2018-07-04	17:00:00
2018-09-03	17:00:00
2018-11-22	17:00:00
2019-01-21	17:00:00
2019-02-18	17:00:00
2019-05-27	17:00:00
2019-07-04	17:00:00
2019-09-02	17:00:00
2019-11-28	17:00:00
2020-01-20	17:00:00
2020-02-17	17:00:00
2020-05-25	17:00:00
2020-09-07	17:00:00
2020-11-26	17:00:00
2021-01-18	17:00:00
2021-02-15	17:00:00
2021-05-31	17:00:00
2021-07-05	17:00:00
2021-09-06	17:00:00
2021-11-25	17:00:00
2022-01-17	17:00:00
2022-02-21	17:00:00

2022-05-30	17:00:00
2022-06-20	17:00:00
2022-07-04	17:00:00
2022-09-05	17:00:00
2022-11-24	17:00:00
2023-01-16	17:00:00
2023-02-20	17:00:00
2023-05-29	17:00:00
2023-06-19	17:00:00
2023-07-04	17:00:00
2023-09-04	17:00:00
2023-11-23	17:00:00
2023-12-25	17:00:00
2024-01-01	17:00:00
2024-01-15	17:00:00
2024-02-19	17:00:00
2024-05-27	17:00:00
2024-06-19	17:00:00
2024-07-04	17:00:00
2024-09-02	17:00:00
2024-11-28	17:00:00
2024-12-25	17:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00

2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The Micro Swiss Franc/U.S. Dollar (CHF/USD) Futures contract in the CME Future market trades in the `America/Chicago` time zone.

CME

NIY

Introduction

This page shows the trading hours, holidays, and time zone of the Nikkei/YEN Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Nikkei/YEN Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Nikkei/YEN Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Nikkei/YEN Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Nikkei/YEN Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Nikkei/YEN Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	11:30:00
2009-02-16	11:30:00
2009-05-25	11:30:00
2009-07-03	11:30:00
2009-09-07	11:30:00
2009-11-26	11:30:00
2009-11-27	13:15:00
2009-12-24	13:15:00
2010-01-18	11:30:00
2010-02-15	11:30:00
2010-04-02	09:15:00
2010-05-31	11:30:00
2010-07-05	11:30:00
2010-09-06	11:30:00
2010-11-25	11:30:00
2010-11-26	13:15:00

2011-01-17	11:30:00
2011-02-21	11:30:00
2011-05-30	11:30:00
2011-07-04	11:30:00
2011-09-05	11:30:00
2011-11-24	11:30:00
2011-11-25	13:15:00
2012-01-16	11:30:00
2012-02-20	11:30:00
2012-04-06	09:15:00
2012-05-28	11:30:00
2012-07-03	13:15:00
2012-07-04	11:30:00
2012-09-03	11:30:00
2012-11-22	11:30:00
2012-11-23	13:15:00
2012-12-24	13:15:00
2013-01-21	11:30:00
2013-02-18	11:30:00
2013-05-27	11:30:00
2013-07-03	13:15:00
2013-07-04	11:30:00
2013-09-02	11:30:00
2013-11-28	11:30:00
2013-11-29	13:15:00
2013-12-24	13:15:00
2014-01-20	11:30:00
2014-02-17	11:30:00

2014-05-26	13:00:00
2014-07-03	13:15:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:15:00
2014-12-24	13:15:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-04-03	09:15:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:15:00
2015-12-24	13:15:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:15:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-03	13:15:00
2017-07-04	13:00:00

2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:15:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-03	13:15:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:15:00
2018-12-24	13:15:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-03	13:15:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:15:00
2019-12-24	13:15:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:15:00

2020-12-24	13:15:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-04-02	09:15:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:15:00
2022-01-17	13:00:00
2022-02-21	13:00:00
2022-05-30	13:00:00
2022-06-20	13:00:00
2022-07-04	13:00:00
2022-09-05	13:00:00
2022-11-24	13:00:00
2022-11-25	13:15:00
2023-01-16	13:00:00
2023-02-20	13:00:00
2023-04-07	09:15:00
2023-05-29	13:00:00
2023-06-19	13:00:00
2023-07-04	13:00:00
2023-09-04	13:00:00
2023-11-23	13:00:00
2023-11-24	13:15:00
2024-01-15	13:00:00
2024-02-19	13:00:00

2024-05-27	13:00:00
2024-06-19	13:00:00
2024-07-04	13:00:00
2024-09-02	13:00:00
2024-11-28	13:00:00
2024-12-24	13:15:00
2024-12-31	17:00:00
2025-01-20	13:00:00
2025-02-17	13:00:00
2025-04-17	17:00:00
2025-05-26	13:00:00
2025-06-19	13:00:00
2025-07-03	13:15:00
2025-07-04	13:00:00
2025-09-01	13:00:00
2025-11-27	13:00:00
2025-11-28	13:15:00
2025-12-24	13:15:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	13:00:00
2026-02-16	13:00:00
2026-04-03	09:15:00
2026-05-25	13:00:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	13:00:00
2026-11-26	13:00:00

2026-11-27	13:15:00
2026-12-24	13:15:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Nikkei/YEN Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00

2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-07-03	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Nikkei/YEN Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

NKD

Introduction

This page shows the trading hours, holidays, and time zone of the Nikkei/USD Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the Nikkei/USD Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Nikkei/USD Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Nikkei/USD Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Nikkei/USD Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Nikkei/USD Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	11:30:00
2009-02-16	11:30:00
2009-05-25	11:30:00
2009-07-03	11:30:00
2009-09-07	11:30:00
2009-11-26	11:30:00
2009-11-27	13:15:00
2009-12-24	13:15:00
2010-01-18	11:30:00
2010-02-15	11:30:00
2010-04-02	09:15:00
2010-05-31	11:30:00
2010-07-05	11:30:00
2010-09-06	11:30:00
2010-11-25	11:30:00
2010-11-26	13:15:00

2011-01-17	11:30:00
2011-02-21	11:30:00
2011-05-30	11:30:00
2011-07-04	11:30:00
2011-09-05	11:30:00
2011-11-24	11:30:00
2011-11-25	13:15:00
2012-01-16	11:30:00
2012-02-20	11:30:00
2012-04-06	09:15:00
2012-05-28	11:30:00
2012-07-03	13:15:00
2012-07-04	11:30:00
2012-09-03	11:30:00
2012-11-22	11:30:00
2012-11-23	13:15:00
2012-12-24	13:15:00
2013-01-21	11:30:00
2013-02-18	11:30:00
2013-05-27	11:30:00
2013-07-03	13:15:00
2013-07-04	11:30:00
2013-09-02	11:30:00
2013-11-28	11:30:00
2013-11-29	13:15:00
2013-12-24	13:15:00
2014-01-20	11:30:00
2014-02-17	11:30:00

2014-05-26	13:00:00
2014-07-03	13:15:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:15:00
2014-12-24	13:15:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-04-03	09:15:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:15:00
2015-12-24	13:15:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:15:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-03	13:15:00
2017-07-04	13:00:00

2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:15:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-03	13:15:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:15:00
2018-12-24	13:15:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-03	13:15:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:15:00
2019-12-24	13:15:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:15:00

2020-12-24	13:15:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-04-02	09:15:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:15:00
2022-01-17	13:00:00
2022-02-21	13:00:00
2022-05-30	13:00:00
2022-06-20	13:00:00
2022-07-04	13:00:00
2022-09-05	13:00:00
2022-11-24	13:00:00
2022-11-25	13:15:00
2023-01-16	13:00:00
2023-02-20	13:00:00
2023-04-07	09:15:00
2023-05-29	13:00:00
2023-06-19	13:00:00
2023-07-04	13:00:00
2023-09-04	13:00:00
2023-11-23	13:00:00
2023-11-24	13:15:00
2024-01-15	13:00:00
2024-02-19	13:00:00

2024-05-27	13:00:00
2024-06-19	13:00:00
2024-07-04	13:00:00
2024-09-02	13:00:00
2024-11-28	13:00:00
2024-12-24	13:15:00
2024-12-31	17:00:00
2025-01-20	13:00:00
2025-02-17	13:00:00
2025-04-17	17:00:00
2025-05-26	13:00:00
2025-06-19	13:00:00
2025-07-03	13:15:00
2025-07-04	13:00:00
2025-09-01	13:00:00
2025-11-27	13:00:00
2025-11-28	13:15:00
2025-12-24	13:15:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	13:00:00
2026-02-16	13:00:00
2026-04-03	09:15:00
2026-05-25	13:00:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	13:00:00
2026-11-26	13:00:00

2026-11-27	13:15:00
2026-12-24	13:15:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Nikkei/USD Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00

2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-07-03	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Nikkei/USD Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

NQ

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini Nasdaq-100 Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini Nasdaq-100 Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini Nasdaq-100 Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini Nasdaq-100 Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the E-mini Nasdaq-100 Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the E-mini Nasdaq-100 Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	11:30:00
2009-02-16	11:30:00
2009-05-25	11:30:00
2009-07-03	11:30:00
2009-09-07	11:30:00
2009-11-26	11:30:00
2009-11-27	13:15:00
2009-12-24	13:15:00
2010-01-18	11:30:00
2010-02-15	11:30:00
2010-04-02	09:15:00
2010-05-31	11:30:00
2010-07-05	11:30:00
2010-09-06	11:30:00
2010-11-25	11:30:00
2010-11-26	13:15:00

2011-01-17	11:30:00
2011-02-21	11:30:00
2011-05-30	11:30:00
2011-07-04	11:30:00
2011-09-05	11:30:00
2011-11-24	11:30:00
2011-11-25	13:15:00
2012-01-16	11:30:00
2012-02-20	11:30:00
2012-04-06	09:15:00
2012-05-28	11:30:00
2012-07-03	13:15:00
2012-07-04	11:30:00
2012-09-03	11:30:00
2012-11-22	11:30:00
2012-11-23	13:15:00
2012-12-24	13:15:00
2013-01-21	11:30:00
2013-02-18	11:30:00
2013-05-27	11:30:00
2013-07-03	13:15:00
2013-07-04	11:30:00
2013-09-02	11:30:00
2013-11-28	11:30:00
2013-11-29	13:15:00
2013-12-24	13:15:00
2014-01-20	11:30:00
2014-02-17	11:30:00

2014-05-26	13:00:00
2014-07-03	13:15:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:15:00
2014-12-24	13:15:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-04-03	09:15:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:15:00
2015-12-24	13:15:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:15:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-03	13:15:00
2017-07-04	13:00:00

2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:15:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-03	13:15:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:15:00
2018-12-24	13:15:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-03	13:15:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:15:00
2019-12-24	13:15:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:15:00

2020-12-24	13:15:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-04-02	09:15:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:15:00
2022-01-17	13:00:00
2022-02-21	13:00:00
2022-05-30	13:00:00
2022-06-20	13:00:00
2022-07-04	13:00:00
2022-09-05	13:00:00
2022-11-24	13:00:00
2022-11-25	13:15:00
2023-01-16	13:00:00
2023-02-20	13:00:00
2023-04-07	09:15:00
2023-05-29	13:00:00
2023-06-19	13:00:00
2023-07-04	13:00:00
2023-09-04	13:00:00
2023-11-23	13:00:00
2023-11-24	13:15:00
2024-01-15	13:00:00
2024-02-19	13:00:00

2024-05-27	13:00:00
2024-06-19	13:00:00
2024-07-04	13:00:00
2024-09-02	13:00:00
2024-11-28	13:00:00
2024-12-24	13:15:00
2024-12-31	17:00:00
2025-01-20	13:00:00
2025-02-17	13:00:00
2025-04-17	17:00:00
2025-05-26	13:00:00
2025-06-19	13:00:00
2025-07-03	13:15:00
2025-07-04	13:00:00
2025-09-01	13:00:00
2025-11-27	13:00:00
2025-11-28	13:15:00
2025-12-24	13:15:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	13:00:00
2026-02-16	13:00:00
2026-04-03	09:15:00
2026-05-25	13:00:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	13:00:00
2026-11-26	13:00:00

2026-11-27	13:15:00
2026-12-24	13:15:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the E-mini Nasdaq-100 Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00

2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-07-03	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The E-mini Nasdaq-100 Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

RS1

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini Russell 1000 future contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini Russell 1000 future contract in the CME Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini Russell 1000 future contract in the CME Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini Russell 1000 future contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the E-mini Russell 1000 future contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the E-mini Russell 1000 future contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	11:30:00
2009-02-16	11:30:00
2009-05-25	11:30:00
2009-07-03	11:30:00
2009-09-07	11:30:00
2009-11-26	11:30:00
2009-11-27	13:15:00
2009-12-24	13:15:00
2010-01-18	11:30:00
2010-02-15	11:30:00
2010-04-02	09:15:00
2010-05-31	11:30:00
2010-07-05	11:30:00
2010-09-06	11:30:00
2010-11-25	11:30:00
2010-11-26	13:15:00

2011-01-17	11:30:00
2011-02-21	11:30:00
2011-05-30	11:30:00
2011-07-04	11:30:00
2011-09-05	11:30:00
2011-11-24	11:30:00
2011-11-25	13:15:00
2012-01-16	11:30:00
2012-02-20	11:30:00
2012-04-06	09:15:00
2012-05-28	11:30:00
2012-07-03	13:15:00
2012-07-04	11:30:00
2012-09-03	11:30:00
2012-11-22	11:30:00
2012-11-23	13:15:00
2012-12-24	13:15:00
2013-01-21	11:30:00
2013-02-18	11:30:00
2013-05-27	11:30:00
2013-07-03	13:15:00
2013-07-04	11:30:00
2013-09-02	11:30:00
2013-11-28	11:30:00
2013-11-29	13:15:00
2013-12-24	13:15:00
2014-01-20	11:30:00
2014-02-17	11:30:00

2014-05-26	13:00:00
2014-07-03	13:15:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:15:00
2014-12-24	13:15:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-04-03	09:15:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:15:00
2015-12-24	13:15:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:15:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-03	13:15:00
2017-07-04	13:00:00

2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:15:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-03	13:15:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:15:00
2018-12-24	13:15:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-03	13:15:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:15:00
2019-12-24	13:15:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:15:00

2020-12-24	13:15:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-04-02	09:15:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:15:00
2022-01-17	13:00:00
2022-02-21	13:00:00
2022-05-30	13:00:00
2022-06-20	13:00:00
2022-07-04	13:00:00
2022-09-05	13:00:00
2022-11-24	13:00:00
2022-11-25	13:15:00
2023-01-16	13:00:00
2023-02-20	13:00:00
2023-04-07	09:15:00
2023-05-29	13:00:00
2023-06-19	13:00:00
2023-07-04	13:00:00
2023-09-04	13:00:00
2023-11-23	13:00:00
2023-11-24	13:15:00
2024-01-15	13:00:00
2024-02-19	13:00:00

2024-05-27	13:00:00
2024-06-19	13:00:00
2024-07-04	13:00:00
2024-09-02	13:00:00
2024-11-28	13:00:00
2024-12-24	13:15:00
2024-12-31	17:00:00
2025-01-20	13:00:00
2025-02-17	13:00:00
2025-04-17	17:00:00
2025-05-26	13:00:00
2025-06-19	13:00:00
2025-07-03	13:15:00
2025-07-04	13:00:00
2025-09-01	13:00:00
2025-11-27	13:00:00
2025-11-28	13:15:00
2025-12-24	13:15:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	13:00:00
2026-02-16	13:00:00
2026-04-03	09:15:00
2026-05-25	13:00:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	13:00:00
2026-11-26	13:00:00

2026-11-27	13:15:00
2026-12-24	13:15:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the E-mini Russell 1000 future contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00

2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-07-03	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The E-mini Russell 1000 future contract in the CME Future market trades in the `America/New York` time zone.

CME

RTY

Introduction

This page shows the trading hours, holidays, and time zone of the E-mini Russell 2000 Index Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the E-mini Russell 2000 Index Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the E-mini Russell 2000 Index Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the E-mini Russell 2000 Index Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the E-mini Russell 2000 Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the E-mini Russell 2000 Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	11:30:00
2009-02-16	11:30:00
2009-05-25	11:30:00
2009-07-03	11:30:00
2009-09-07	11:30:00
2009-11-26	11:30:00
2009-11-27	13:15:00
2009-12-24	13:15:00
2010-01-18	11:30:00
2010-02-15	11:30:00
2010-04-02	09:15:00
2010-05-31	11:30:00
2010-07-05	11:30:00
2010-09-06	11:30:00
2010-11-25	11:30:00
2010-11-26	13:15:00

2011-01-17	11:30:00
2011-02-21	11:30:00
2011-05-30	11:30:00
2011-07-04	11:30:00
2011-09-05	11:30:00
2011-11-24	11:30:00
2011-11-25	13:15:00
2012-01-16	11:30:00
2012-02-20	11:30:00
2012-04-06	09:15:00
2012-05-28	11:30:00
2012-07-03	13:15:00
2012-07-04	11:30:00
2012-09-03	11:30:00
2012-11-22	11:30:00
2012-11-23	13:15:00
2012-12-24	13:15:00
2013-01-21	11:30:00
2013-02-18	11:30:00
2013-05-27	11:30:00
2013-07-03	13:15:00
2013-07-04	11:30:00
2013-09-02	11:30:00
2013-11-28	11:30:00
2013-11-29	13:15:00
2013-12-24	13:15:00
2014-01-20	11:30:00
2014-02-17	11:30:00

2014-05-26	13:00:00
2014-07-03	13:15:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:15:00
2014-12-24	13:15:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-04-03	09:15:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:15:00
2015-12-24	13:15:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:15:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-03	13:15:00
2017-07-04	13:00:00

2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:15:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-03	13:15:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:15:00
2018-12-24	13:15:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-03	13:15:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:15:00
2019-12-24	13:15:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:15:00

2020-12-24	13:15:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-04-02	09:15:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:15:00
2022-01-17	13:00:00
2022-02-21	13:00:00
2022-05-30	13:00:00
2022-06-20	13:00:00
2022-07-04	13:00:00
2022-09-05	13:00:00
2022-11-24	13:00:00
2022-11-25	13:15:00
2023-01-16	13:00:00
2023-02-20	13:00:00
2023-04-07	09:15:00
2023-05-29	13:00:00
2023-06-19	13:00:00
2023-07-04	13:00:00
2023-09-04	13:00:00
2023-11-23	13:00:00
2023-11-24	13:15:00
2024-01-15	13:00:00
2024-02-19	13:00:00

2024-05-27	13:00:00
2024-06-19	13:00:00
2024-07-04	13:00:00
2024-09-02	13:00:00
2024-11-28	13:00:00
2024-12-24	13:15:00
2024-12-31	17:00:00
2025-01-20	13:00:00
2025-02-17	13:00:00
2025-04-17	17:00:00
2025-05-26	13:00:00
2025-06-19	13:00:00
2025-07-03	13:15:00
2025-07-04	13:00:00
2025-09-01	13:00:00
2025-11-27	13:00:00
2025-11-28	13:15:00
2025-12-24	13:15:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	13:00:00
2026-02-16	13:00:00
2026-04-03	09:15:00
2026-05-25	13:00:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	13:00:00
2026-11-26	13:00:00

2026-11-27	13:15:00
2026-12-24	13:15:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the E-mini Russell 2000 Index Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00

2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-07-03	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The E-mini Russell 2000 Index Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

RX

Introduction

This page shows the trading hours, holidays, and time zone of the DJRE Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the DJRE Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the DJRE Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the DJRE Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the CME Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04

2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-04-10
2009-12-25	2010-01-01	2010-12-24	2011-01-01	2011-04-22
2011-12-26	2012-01-02	2012-10-29	2012-10-30	2012-12-25
2013-01-01	2013-03-29	2013-12-25	2014-01-01	2014-04-18
2014-12-25	2015-01-01	2015-12-25	2016-01-01	2016-03-25
2016-12-25	2016-12-26	2017-01-02	2017-04-14	2017-12-25
2018-01-01	2018-03-30	2018-12-05	2018-12-25	2019-01-01
2019-04-19	2019-12-25	2020-01-01	2020-04-10	2020-12-25
2021-01-01	2021-12-24	2022-01-01	2022-04-15	2022-12-26
2023-01-02	2023-12-24	2023-12-31	2024-03-29	2025-01-20
2025-04-18				

Early Closes

The following table shows the early closes for the DJRE Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/New York)**
2009-01-19	11:30:00
2009-02-16	11:30:00
2009-05-25	11:30:00
2009-07-03	11:30:00
2009-09-07	11:30:00
2009-11-26	11:30:00
2009-11-27	13:15:00
2009-12-24	13:15:00
2010-01-18	11:30:00
2010-02-15	11:30:00
2010-04-02	09:15:00
2010-05-31	11:30:00
2010-07-05	11:30:00
2010-09-06	11:30:00

2010-11-25	11:30:00
2010-11-26	13:15:00
2011-01-17	11:30:00
2011-02-21	11:30:00
2011-05-30	11:30:00
2011-07-04	11:30:00
2011-09-05	11:30:00
2011-11-24	11:30:00
2011-11-25	13:15:00
2012-01-16	11:30:00
2012-02-20	11:30:00
2012-04-06	09:15:00
2012-05-28	11:30:00
2012-07-03	13:15:00
2012-07-04	11:30:00
2012-09-03	11:30:00
2012-11-22	11:30:00
2012-11-23	13:15:00
2012-12-24	13:15:00
2013-01-21	11:30:00
2013-02-18	11:30:00
2013-05-27	11:30:00
2013-07-03	13:15:00
2013-07-04	11:30:00
2013-09-02	11:30:00
2013-11-28	11:30:00
2013-11-29	13:15:00
2013-12-24	13:15:00

2014-01-20	11:30:00
2014-02-17	11:30:00
2014-05-26	13:00:00
2014-07-03	13:15:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:15:00
2014-12-24	13:15:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-04-03	09:15:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:15:00
2015-12-24	13:15:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:15:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00

2017-07-03	13:15:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:15:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-03	13:15:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:15:00
2018-12-24	13:15:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-03	13:15:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:15:00
2019-12-24	13:15:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00

2020-11-26	13:00:00
2020-11-27	13:15:00
2020-12-24	13:15:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-04-02	09:15:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:15:00
2022-01-17	13:00:00
2022-02-21	13:00:00
2022-05-30	13:00:00
2022-06-20	13:00:00
2022-07-04	13:00:00
2022-09-05	13:00:00
2022-11-24	13:00:00
2022-11-25	13:15:00
2023-01-16	13:00:00
2023-02-20	13:00:00
2023-04-07	09:15:00
2023-05-29	13:00:00
2023-06-19	13:00:00
2023-07-04	13:00:00
2023-09-04	13:00:00
2023-11-23	13:00:00
2023-11-24	13:15:00

Late Opens

The following table shows the late opens for the DJRE Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00

2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00

2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00

2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00

Time Zone

The DJRE Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

SDA

Introduction

This page shows the trading hours, holidays, and time zone of the SP500 S&P 500 Annual Dividend Index future contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the SP500 S&P 500 Annual Dividend Index future contract in the CME Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the SP500 S&P 500 Annual Dividend Index future contract in the CME Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the SP500 S&P 500 Annual Dividend Index future contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the SP500 S&P 500 Annual Dividend Index future contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the SP500 S&P 500 Annual Dividend Index future contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	11:30:00
2009-02-16	11:30:00
2009-05-25	11:30:00
2009-07-03	11:30:00
2009-09-07	11:30:00
2009-11-26	11:30:00
2009-11-27	13:15:00
2009-12-24	13:15:00
2010-01-18	11:30:00
2010-02-15	11:30:00
2010-04-02	09:15:00
2010-05-31	11:30:00
2010-07-05	11:30:00
2010-09-06	11:30:00
2010-11-25	11:30:00

2010-11-26	13:15:00
2011-01-17	11:30:00
2011-02-21	11:30:00
2011-05-30	11:30:00
2011-07-04	11:30:00
2011-09-05	11:30:00
2011-11-24	11:30:00
2011-11-25	13:15:00
2012-01-16	11:30:00
2012-02-20	11:30:00
2012-04-06	09:15:00
2012-05-28	11:30:00
2012-07-03	13:15:00
2012-07-04	11:30:00
2012-09-03	11:30:00
2012-11-22	11:30:00
2012-11-23	13:15:00
2012-12-24	13:15:00
2013-01-21	11:30:00
2013-02-18	11:30:00
2013-05-27	11:30:00
2013-07-03	13:15:00
2013-07-04	11:30:00
2013-09-02	11:30:00
2013-11-28	11:30:00
2013-11-29	13:15:00
2013-12-24	13:15:00
2014-01-20	11:30:00

2014-02-17	11:30:00
2014-05-26	13:00:00
2014-07-03	13:15:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:15:00
2014-12-24	13:15:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-04-03	09:15:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:15:00
2015-12-24	13:15:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:15:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-03	13:15:00

2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:15:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-03	13:15:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:15:00
2018-12-24	13:15:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-03	13:15:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:15:00
2019-12-24	13:15:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00
2020-11-26	13:00:00

2020-11-27	13:15:00
2020-12-24	13:15:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-04-02	09:15:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:15:00
2022-01-17	13:00:00
2022-02-21	13:00:00
2022-05-30	13:00:00
2022-06-20	13:00:00
2022-07-04	13:00:00
2022-09-05	13:00:00
2022-11-24	13:00:00
2022-11-25	13:15:00
2023-01-16	13:00:00
2023-02-20	13:00:00
2023-04-07	09:15:00
2023-05-29	13:00:00
2023-06-19	13:00:00
2023-07-04	13:00:00
2023-09-04	13:00:00
2023-11-23	13:00:00
2023-11-24	13:15:00
2024-01-15	13:00:00

2024-02-19	13:00:00
2024-05-27	13:00:00
2024-06-19	13:00:00
2024-07-04	13:00:00
2024-09-02	13:00:00
2024-11-28	13:00:00
2024-12-24	13:15:00
2024-12-31	17:00:00
2025-01-20	13:00:00
2025-02-17	13:00:00
2025-04-17	17:00:00
2025-05-26	13:00:00
2025-06-19	13:00:00
2025-07-03	13:15:00
2025-07-04	13:00:00
2025-09-01	13:00:00
2025-11-27	13:00:00
2025-11-28	13:15:00
2025-12-24	13:15:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	13:00:00
2026-02-16	13:00:00
2026-04-03	09:15:00
2026-05-25	13:00:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	13:00:00

2026-11-26	13:00:00
2026-11-27	13:15:00
2026-12-24	13:15:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the SP500 S&P 500 Annual Dividend Index future contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00

2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-07-03	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The SP500 S&P 500 Annual Dividend Index future contract in the CME Future market trades in the `America/New York` time zone.

CME

TPD

Introduction

This page shows the trading hours, holidays, and time zone of the TOPIX USD Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the TOPIX USD Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the TOPIX USD Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the TOPIX USD Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the TOPIX USD Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the TOPIX USD Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	11:30:00
2009-02-16	11:30:00
2009-05-25	11:30:00
2009-07-03	11:30:00
2009-09-07	11:30:00
2009-11-26	11:30:00
2009-11-27	13:15:00
2009-12-24	13:15:00
2010-01-18	11:30:00
2010-02-15	11:30:00
2010-04-02	09:15:00
2010-05-31	11:30:00
2010-07-05	11:30:00
2010-09-06	11:30:00
2010-11-25	11:30:00
2010-11-26	13:15:00

2011-01-17	11:30:00
2011-02-21	11:30:00
2011-05-30	11:30:00
2011-07-04	11:30:00
2011-09-05	11:30:00
2011-11-24	11:30:00
2011-11-25	13:15:00
2012-01-16	11:30:00
2012-02-20	11:30:00
2012-04-06	09:15:00
2012-05-28	11:30:00
2012-07-03	13:15:00
2012-07-04	11:30:00
2012-09-03	11:30:00
2012-11-22	11:30:00
2012-11-23	13:15:00
2012-12-24	13:15:00
2013-01-21	11:30:00
2013-02-18	11:30:00
2013-05-27	11:30:00
2013-07-03	13:15:00
2013-07-04	11:30:00
2013-09-02	11:30:00
2013-11-28	11:30:00
2013-11-29	13:15:00
2013-12-24	13:15:00
2014-01-20	11:30:00
2014-02-17	11:30:00

2014-05-26	13:00:00
2014-07-03	13:15:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:15:00
2014-12-24	13:15:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-04-03	09:15:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:15:00
2015-12-24	13:15:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:15:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-03	13:15:00
2017-07-04	13:00:00

2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:15:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-03	13:15:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:15:00
2018-12-24	13:15:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-03	13:15:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:15:00
2019-12-24	13:15:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:15:00

2020-12-24	13:15:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-04-02	09:15:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:15:00
2022-01-17	13:00:00
2022-02-21	13:00:00
2022-05-30	13:00:00
2022-06-20	13:00:00
2022-07-04	13:00:00
2022-09-05	13:00:00
2022-11-24	13:00:00
2022-11-25	13:15:00
2023-01-16	13:00:00
2023-02-20	13:00:00
2023-04-07	09:15:00
2023-05-29	13:00:00
2023-06-19	13:00:00
2023-07-04	13:00:00
2023-09-04	13:00:00
2023-11-23	13:00:00
2023-11-24	13:15:00
2024-01-15	13:00:00
2024-02-19	13:00:00

2024-05-27	13:00:00
2024-06-19	13:00:00
2024-07-04	13:00:00
2024-09-02	13:00:00
2024-11-28	13:00:00
2024-12-24	13:15:00
2024-12-31	17:00:00
2025-01-20	13:00:00
2025-02-17	13:00:00
2025-04-17	17:00:00
2025-05-26	13:00:00
2025-06-19	13:00:00
2025-07-03	13:15:00
2025-07-04	13:00:00
2025-09-01	13:00:00
2025-11-27	13:00:00
2025-11-28	13:15:00
2025-12-24	13:15:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	13:00:00
2026-02-16	13:00:00
2026-04-03	09:15:00
2026-05-25	13:00:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	13:00:00
2026-11-26	13:00:00

2026-11-27	13:15:00
2026-12-24	13:15:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the TOPIX USD Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00

2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-07-03	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The TOPIX USD Futures contract in the CME Future market trades in the `America/New York` time zone.

CME

TPY

Introduction

This page shows the trading hours, holidays, and time zone of the TOPIX JPY Futures contract in the CME Future market.

Pre-market Hours

The following table shows the pre-market hours for the TOPIX JPY Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the TOPIX JPY Futures contract in the CME Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the TOPIX JPY Futures contract in the CME Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the TOPIX JPY Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	
2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the TOPIX JPY Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	11:30:00
2009-02-16	11:30:00
2009-05-25	11:30:00
2009-07-03	11:30:00
2009-09-07	11:30:00
2009-11-26	11:30:00
2009-11-27	13:15:00
2009-12-24	13:15:00
2010-01-18	11:30:00
2010-02-15	11:30:00
2010-04-02	09:15:00
2010-05-31	11:30:00
2010-07-05	11:30:00
2010-09-06	11:30:00
2010-11-25	11:30:00
2010-11-26	13:15:00

2011-01-17	11:30:00
2011-02-21	11:30:00
2011-05-30	11:30:00
2011-07-04	11:30:00
2011-09-05	11:30:00
2011-11-24	11:30:00
2011-11-25	13:15:00
2012-01-16	11:30:00
2012-02-20	11:30:00
2012-04-06	09:15:00
2012-05-28	11:30:00
2012-07-03	13:15:00
2012-07-04	11:30:00
2012-09-03	11:30:00
2012-11-22	11:30:00
2012-11-23	13:15:00
2012-12-24	13:15:00
2013-01-21	11:30:00
2013-02-18	11:30:00
2013-05-27	11:30:00
2013-07-03	13:15:00
2013-07-04	11:30:00
2013-09-02	11:30:00
2013-11-28	11:30:00
2013-11-29	13:15:00
2013-12-24	13:15:00
2014-01-20	11:30:00
2014-02-17	11:30:00

2014-05-26	13:00:00
2014-07-03	13:15:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:15:00
2014-12-24	13:15:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-04-03	09:15:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:15:00
2015-12-24	13:15:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:15:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-03	13:15:00
2017-07-04	13:00:00

2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:15:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-03	13:15:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:15:00
2018-12-24	13:15:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-03	13:15:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:15:00
2019-12-24	13:15:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:15:00

2020-12-24	13:15:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-04-02	09:15:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:15:00
2022-01-17	13:00:00
2022-02-21	13:00:00
2022-05-30	13:00:00
2022-06-20	13:00:00
2022-07-04	13:00:00
2022-09-05	13:00:00
2022-11-24	13:00:00
2022-11-25	13:15:00
2023-01-16	13:00:00
2023-02-20	13:00:00
2023-04-07	09:15:00
2023-05-29	13:00:00
2023-06-19	13:00:00
2023-07-04	13:00:00
2023-09-04	13:00:00
2023-11-23	13:00:00
2023-11-24	13:15:00
2024-01-15	13:00:00
2024-02-19	13:00:00

2024-05-27	13:00:00
2024-06-19	13:00:00
2024-07-04	13:00:00
2024-09-02	13:00:00
2024-11-28	13:00:00
2024-12-24	13:15:00
2024-12-31	17:00:00
2025-01-20	13:00:00
2025-02-17	13:00:00
2025-04-17	17:00:00
2025-05-26	13:00:00
2025-06-19	13:00:00
2025-07-03	13:15:00
2025-07-04	13:00:00
2025-09-01	13:00:00
2025-11-27	13:00:00
2025-11-28	13:15:00
2025-12-24	13:15:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	13:00:00
2026-02-16	13:00:00
2026-04-03	09:15:00
2026-05-25	13:00:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	13:00:00
2026-11-26	13:00:00

2026-11-27	13:15:00
2026-12-24	13:15:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the TOPIX JPY Futures contract in the CME Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00

2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-07-03	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The TOPIX JPY Futures contract in the CME Future market trades in the `America/New York` time zone.

Market Hours

COMEX

Introduction

This page shows the trading hours, holidays, and time zone of the COMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the COMEX Future market:

Weekday	Time (America/Chicago)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the COMEX Future market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the COMEX Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the COMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04

2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19

2023-07-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-09	2025-01-20
2025-04-18				

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The COMEX Future market trades in the following time zones:

- `America/Chicago`
- `America/New York`

Assets With Other Hours

The following table shows the contracts that have different trading periods than the overall COMEX Future market:

Symbol	Name
AUP	Aluminum MW U.S. Transaction Premium Platts (25MT) Futures
EDP	Aluminium European Premium Duty-Paid (Metal Bulletin) Futures
GC	Gold Futures
HG	Copper Futures
MGC	Micro Gold Futures
MGT	Micro Gold TAS Futures
SI	Silver Futures
SIL	Micro Silver Futures

COMEX

AUP

Introduction

This page shows the trading hours, holidays, and time zone of the Aluminum MW U.S. Transaction Premium Platts (25MT) Futures contract in the COMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Aluminum MW U.S. Transaction Premium Platts (25MT) Futures contract in the COMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Aluminum MW U.S. Transaction Premium Platts (25MT) Futures contract in the COMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Aluminum MW U.S. Transaction Premium Platts (25MT) Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Aluminum MW U.S. Transaction Premium Platts (25MT) Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Aluminum MW U.S. Transaction Premium Platts (25MT) Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00

2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Aluminum MW U.S. Transaction Premium Platts (25MT) Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00

2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00

2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00

2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00

2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Aluminum MW U.S. Transaction Premium Platts (25MT) Futures contract in the COMEX Future market trades in the `America/New York` time zone.

COMEX

EDP

Introduction

This page shows the trading hours, holidays, and time zone of the Aluminium European Premium Duty-Paid (Metal Bulletin) Futures contract in the COMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Aluminium European Premium Duty-Paid (Metal Bulletin) Futures contract in the COMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Aluminium European Premium Duty-Paid (Metal Bulletin) Futures contract in the COMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Aluminium European Premium Duty-Paid (Metal Bulletin) Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Aluminium European Premium Duty-Paid (Metal Bulletin) Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Aluminium European Premium Duty-Paid (Metal Bulletin) Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00

2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Aluminium European Premium Duty-Paid (Metal Bulletin) Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00

2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00

2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00

2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00

2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Aluminium European Premium Duty-Paid (Metal Bulletin) Futures contract in the COMEX Future market trades in the `America/New York` time zone.

COMEX

GC

Introduction

This page shows the trading hours, holidays, and time zone of the Gold Futures contract in the COMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Gold Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Gold Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Gold Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Gold Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Gold Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2024-01-15	14:30:00
2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00

2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Gold Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Gold Futures contract in the COMEX Future market trades in the `America/New York` time zone.

COMEX

HG

Introduction

This page shows the trading hours, holidays, and time zone of the Copper Futures contract in the COMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Copper Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Copper Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Copper Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Copper Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Copper Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00
2010-05-31	13:15:00

2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00

2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00

2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2024-01-15	14:30:00
2024-02-19	14:30:00
2024-03-28	17:00:00

2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00

2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Copper Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Copper Futures contract in the COMEX Future market trades in the `America/New York` time zone.

COMEX

MGC

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Gold Futures contract in the COMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Gold Futures contract in the COMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Gold Futures contract in the COMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Gold Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro Gold Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro Gold Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00

2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00

2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00

2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00

2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00
2024-02-19	14:30:00

2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00
2026-11-27	14:45:00

2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Micro Gold Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00

2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00

2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00

2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00

2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Micro Gold Futures contract in the COMEX Future market trades in the `America/New York` time zone.

COMEX

MGT

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Gold TAS Futures contract in the COMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Gold TAS Futures contract in the COMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Gold TAS Futures contract in the COMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Gold TAS Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro Gold TAS Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro Gold TAS Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00

2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00

2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00

2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00

2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00
2024-02-19	14:30:00

2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00
2026-11-27	14:45:00

2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Micro Gold TAS Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00

2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00

2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00

2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00

2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Micro Gold TAS Futures contract in the COMEX Future market trades in the `America/New York` time zone.

COMEX

SI

Introduction

This page shows the trading hours, holidays, and time zone of the Silver Futures contract in the COMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Silver Futures contract in the COMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Silver Futures contract in the COMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Silver Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Silver Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Silver Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2024-01-15	14:30:00
2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00

2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Silver Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Silver Futures contract in the COMEX Future market trades in the `America/New York` time zone.

COMEX

SIL

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Silver Futures contract in the COMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Silver Futures contract in the COMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Silver Futures contract in the COMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Silver Futures contract in the COMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro Silver Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro Silver Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00

2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00

2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00

2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00

2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00
2024-02-19	14:30:00

2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00
2026-11-27	14:45:00

2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Micro Silver Futures contract in the COMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00

2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00

2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00

2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00

2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Micro Silver Futures contract in the COMEX Future market trades in the `America/New York` time zone.

Market Hours

EUREX

Introduction

This page shows the trading hours, holidays, and time zone of the EUREX Future market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUREX Future market:

Weekday	**Time (Europe/Berlin)**
Monday	01:10:00 to 22:00:00
Tuesday	01:10:00 to 22:00:00
Wednesday	01:10:00 to 22:00:00
Thursday	01:10:00 to 22:00:00
Friday	01:10:00 to 22:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the EUREX Future market:

Date (*yyyy-mm-dd*)				
2000-04-21	2000-04-24	2000-05-01	2000-12-25	2000-12-26
2001-01-01	2001-04-13	2001-04-16	2001-05-01	2001-12-24
2001-12-25	2001-12-26	2001-12-31	2002-01-01	2002-03-29
2002-04-01	2002-05-01	2002-12-24	2002-12-25	2002-12-26
2002-12-31	2003-01-01	2003-04-18	2003-04-21	2004-01-01
2004-04-09	2004-04-12	2004-05-01	2004-12-24	2004-12-25
2004-12-26	2004-12-31	2005-03-25	2005-03-28	2005-12-26
2006-04-14	2006-04-17	2006-05-01	2006-12-25	2006-12-26

2007-01-01	2007-04-06	2007-04-09	2007-05-01	2007-12-24
2007-12-25	2007-12-26	2007-12-31	2008-01-01	2008-03-21
2008-03-24	2008-05-01	2008-12-24	2008-12-25	2008-12-26
2008-12-31	2009-01-01	2009-04-10	2009-04-13	2009-05-01
2009-12-24	2009-12-25	2009-12-31	2010-01-01	2010-04-02
2010-04-05	2010-12-24	2010-12-31	2011-04-22	2011-04-25
2011-12-26	2012-04-06	2012-04-09	2012-05-01	2012-12-24
2012-12-25	2012-12-26	2012-12-31	2013-01-01	2013-03-29
2013-04-01	2013-05-01	2013-12-24	2013-12-25	2013-12-26
2013-12-31	2014-01-01	2014-04-18	2014-04-21	2014-05-01
2014-12-24	2014-12-25	2014-12-26	2014-12-31	2015-01-01
2015-04-03	2015-04-06	2015-05-01	2015-05-25	2015-12-24
2015-12-25	2015-12-26	2015-12-31	2016-01-01	2016-03-25
2016-03-28	2016-12-26	2017-04-14	2017-04-17	2017-05-01
2017-12-25	2017-12-26	2018-01-01	2018-03-30	2018-04-02
2018-05-01	2018-12-24	2018-12-25	2018-12-26	2018-12-31
2019-01-01	2019-04-19	2019-04-22	2019-05-01	2019-12-24
2019-12-25	2019-12-26	2019-12-31	2020-01-01	2020-04-10
2020-04-13	2020-05-01	2020-12-24	2020-12-25	2020-12-31
2021-01-01	2021-04-02	2021-04-05	2021-12-24	2021-12-31
2022-04-15	2022-04-18	2022-12-26	2023-04-07	2023-04-10
2023-05-01	2023-12-25	2023-12-26	2024-01-01	2024-03-29
2024-04-01	2024-05-01	2024-12-24	2024-12-25	2024-12-26
2024-12-31	2025-01-01	2025-04-18	2025-04-21	2025-05-01
2025-12-24	2025-12-25	2025-12-26	2025-12-31	2026-01-01
2026-04-03	2026-04-06	2026-05-01	2026-12-24	2026-12-25
2026-12-31	2027-01-01			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUREX Future market trades in the `Europe/Berlin` time zone.

Market Hours

HKFE

Introduction

This page shows the trading hours, holidays, and time zone of the HKFE Future market.

Pre-market Hours

The following table shows the pre-market hours for the HKFE Future market:

Weekday	Time (Asia/Hong Kong)
Monday	08:45:00 to 09:15:00, 12:30:00 to 13:00:00
Tuesday	00:00:00 to 03:00:00, 08:45:00 to 09:15:00, 12:30:00 to 13:00:00
Wednesday	00:00:00 to 03:00:00, 08:45:00 to 09:15:00, 12:30:00 to 13:00:00
Thursday	00:00:00 to 03:00:00, 08:45:00 to 09:15:00, 12:30:00 to 13:00:00
Friday	00:00:00 to 03:00:00, 08:45:00 to 09:15:00, 12:30:00 to 13:00:00
Saturday	00:00:00 to 03:00:00

Regular Trading Hours

The following table shows the regular trading hours for the HKFE Future market:

Weekday	Time (Asia/Hong Kong)
Monday	09:15:00 to 12:00:00, 13:00:00 to 16:30:00
Tuesday	09:15:00 to 12:00:00, 13:00:00 to 16:30:00
Wednesday	09:15:00 to 12:00:00, 13:00:00 to 16:30:00
Thursday	09:15:00 to 12:00:00, 13:00:00 to 16:30:00
Friday	09:15:00 to 12:00:00, 13:00:00 to 16:30:00

Post-market Hours

The following table shows the post-market hours for the HKFE Future market:

Weekday	Time (Asia/Hong Kong)
Monday	17:15:00 to 24:00:00
Tuesday	17:15:00 to 24:00:00
Wednesday	17:15:00 to 24:00:00
Thursday	17:15:00 to 24:00:00
Friday	17:15:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the HKFE Future market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-30	2017-01-31	2017-04-04	2017-04-14
2017-04-17	2017-05-01	2017-05-03	2017-05-30	2017-10-02
2017-10-05	2017-12-25	2017-12-26	2018-01-01	2018-02-16
2018-02-19	2018-03-30	2018-04-02	2018-04-05	2018-05-01
2018-05-22	2018-06-18	2018-07-02	2018-09-25	2018-10-01
2018-10-17	2018-12-25	2018-12-26	2019-01-01	2019-02-05
2019-02-06	2019-02-07	2019-04-05	2019-04-19	2019-04-22
2019-05-01	2019-05-13	2019-06-07	2019-07-01	2019-10-01
2019-10-07	2019-12-25	2019-12-26	2020-01-01	2020-01-27
2020-01-28	2020-04-10	2020-04-13	2020-04-30	2020-05-01
2020-06-25	2020-07-01	2020-10-01	2020-10-02	2020-10-26
2020-12-25	2021-01-01	2021-02-12	2021-02-15	2021-04-02
2021-04-05	2021-04-06	2021-05-19	2021-06-14	2021-07-01
2021-09-22	2021-10-01	2021-10-14	2021-12-27	2022-02-01
2022-02-02	2022-02-03	2022-04-05	2022-04-15	2022-04-18
2022-05-02	2022-05-09	2022-06-03	2022-07-01	2022-09-12
2022-10-04	2022-12-26	2022-12-27	2023-01-02	2023-01-23
2023-01-24	2023-01-25	2023-04-05	2023-04-07	2023-04-10
2023-05-01	2023-05-26	2023-06-22	2023-10-02	2023-10-23

2023-12-25	2023-12-26	2024-01-01	2024-02-12	2024-02-13
2024-03-29	2024-04-01	2024-04-04	2024-05-01	2024-05-15
2024-06-10	2024-07-01	2024-09-18	2024-10-01	2024-10-11
2024-12-25	2024-12-26	2025-01-01	2025-01-29	2025-01-30
2025-01-31	2025-04-04	2025-04-18	2025-04-21	2025-05-01
2025-05-05	2025-07-01	2025-10-01	2025-10-07	2025-10-29
2025-12-25	2025-12-26	2026-01-01	2026-02-17	2026-02-18
2026-02-19	2026-04-03	2026-04-06	2026-04-07	2026-05-01
2026-05-25	2026-06-19	2026-07-01	2026-10-01	2026-10-19
2026-12-25				

Early Closes

The following table shows the early closes for the HKFE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (Asia/Hong Kong)
2018-02-15	12:00:00
2018-12-24	12:00:00
2018-12-31	12:00:00
2019-02-04	12:00:00
2019-12-24	12:00:00
2019-12-31	12:00:00
2020-01-24	12:00:00
2020-12-24	12:00:00
2020-12-31	12:00:00
2021-02-11	12:00:00
2021-12-24	12:00:00
2021-12-31	12:00:00
2022-01-31	12:00:00
2024-02-09	12:00:00
2024-12-24	12:00:00
2024-12-31	12:00:00
2025-01-28	12:00:00
2025-12-24	12:00:00
2025-12-31	12:00:00
2026-02-16	12:00:00
2026-12-24	12:00:00
2026-12-31	12:00:00

Late Opens

There are no days with late opens.

Time Zone

The HKFE Future market trades in the `Asia/Hong Kong` time zone.

Assets With Other Hours

The following table shows the contracts that have different trading periods than the overall HKFE Future market:

Symbol	Name
HSI	Hang Seng Index

HKFE

HSI

Introduction

This page shows the trading hours, holidays, and time zone of the Hang Seng Index contract in the HKFE Future market.

Pre-market Hours

The following table shows the pre-market hours for the Hang Seng Index contract in the HKFE Future market:

Weekday	**Time (Asia/Hong Kong)**
Monday	08:45:00 to 09:15:00, 12:30:00 to 13:00:00
Tuesday	00:00:00 to 03:00:00, 08:45:00 to 09:15:00, 12:30:00 to 13:00:00
Wednesday	00:00:00 to 03:00:00, 08:45:00 to 09:15:00, 12:30:00 to 13:00:00
Thursday	00:00:00 to 03:00:00, 08:45:00 to 09:15:00, 12:30:00 to 13:00:00
Friday	00:00:00 to 03:00:00, 08:45:00 to 09:15:00, 12:30:00 to 13:00:00
Saturday	00:00:00 to 03:00:00

Regular Trading Hours

The following table shows the regular trading hours for the Hang Seng Index contract in the HKFE Future market:

Weekday	**Time (Asia/Hong Kong)**
Monday	09:15:00 to 12:00:00, 13:00:00 to 16:30:00
Tuesday	09:15:00 to 12:00:00, 13:00:00 to 16:30:00
Wednesday	09:15:00 to 12:00:00, 13:00:00 to 16:30:00
Thursday	09:15:00 to 12:00:00, 13:00:00 to 16:30:00
Friday	09:15:00 to 12:00:00, 13:00:00 to 16:30:00

Post-market Hours

The following table shows the post-market hours for the Hang Seng Index contract in the HKFE Future market:

Weekday	Time (Asia/Hong Kong)
Monday	17:15:00 to 24:00:00
Tuesday	17:15:00 to 24:00:00
Wednesday	17:15:00 to 24:00:00
Thursday	17:15:00 to 24:00:00
Friday	17:15:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the HKFE Future market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-30	2017-01-31	2017-04-04	2017-04-14
2017-04-17	2017-05-01	2017-05-03	2017-05-30	2017-10-02
2017-10-05	2017-12-25	2017-12-26	2018-01-01	2018-02-16
2018-02-19	2018-03-30	2018-04-02	2018-04-05	2018-05-01
2018-05-22	2018-06-18	2018-07-02	2018-09-25	2018-10-01
2018-10-17	2018-12-25	2018-12-26	2019-01-01	2019-02-05
2019-02-06	2019-02-07	2019-04-05	2019-04-19	2019-04-22
2019-05-01	2019-05-13	2019-06-07	2019-07-01	2019-10-01
2019-10-07	2019-12-25	2019-12-26	2020-01-01	2020-01-27
2020-01-28	2020-04-10	2020-04-13	2020-04-30	2020-05-01
2020-06-25	2020-07-01	2020-10-01	2020-10-02	2020-10-26
2020-12-25	2021-01-01	2021-02-12	2021-02-15	2021-04-02
2021-04-05	2021-04-06	2021-05-19	2021-06-14	2021-07-01
2021-09-22	2021-10-01	2021-10-14	2021-12-27	2022-02-01
2022-02-02	2022-02-03	2022-04-05	2022-04-15	2022-04-18
2022-05-02	2022-05-09	2022-06-03	2022-07-01	2022-09-12
2022-10-04	2022-12-26	2022-12-27	2023-01-02	2023-01-23
2023-01-24	2023-01-25	2023-04-05	2023-04-07	2023-04-10
2023-05-01	2023-05-26	2023-06-22	2023-10-02	2023-10-23

2023-12-25	2023-12-26	2024-01-01	2024-02-12	2024-02-13
2024-03-29	2024-04-01	2024-04-04	2024-05-01	2024-05-15
2024-06-10	2024-07-01	2024-09-18	2024-10-01	2024-10-11
2024-12-25	2024-12-26	2025-01-01	2025-01-29	2025-01-30
2025-01-31	2025-04-04	2025-04-18	2025-04-21	2025-05-01
2025-05-05	2025-07-01	2025-10-01	2025-10-07	2025-10-29
2025-12-25	2025-12-26	2026-01-01	2026-02-17	2026-02-18
2026-02-19	2026-04-03	2026-04-06	2026-04-07	2026-05-01
2026-05-25	2026-06-19	2026-07-01	2026-10-01	2026-10-19
2026-12-25				

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Hang Seng Index contract in the HKFE Future market trades in the `Asia/Hong Kong` time zone.

Market Hours

ICE

Introduction

This page shows the trading hours, holidays, and time zone of the ICE Future market.

Historical data for backtesting is unavailable for ICE. In live trading, LEAN sources this data from your brokerage or a third-party data provider .

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the ICE Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 24:00:00
Tuesday	00:00:00 to 24:00:00
Wednesday	00:00:00 to 24:00:00
Thursday	00:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the ICE Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23

2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03

2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19
2024-07-04	2024-09-02	2024-11-28	2024-12-25	2025-01-01
2025-01-20	2025-04-18	2025-12-25	2026-01-01	2026-12-25
2027-01-01				

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The ICE Future market trades in the `America/New York` time zone.

Assets With Other Hours

The following table shows the contracts that have different trading periods than the overall ICE Future market:

Symbol	Name
B	Brent Crude Futures
CC	Cocoa Futures
CT	Cotton No. 2 Futures
DX	US Dollar Index Futures
G	Low Sulfur Gasoil
KC	Coffee C Arabica Futures
OJ	Frozen Concentrated Orange Juice
SB	Sugar No. 11 Futures

ICE

B

Introduction

This page shows the trading hours, holidays, and time zone of the Brent Crude Futures contract in the ICE Future market.

Historical data for backtesting is unavailable for ICE. In live trading, LEAN sources this data from your brokerage or a third-party data provider .

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Brent Crude Futures contract in the ICE Future market:

Weekday	**Time (America/New York)**
Sunday	20:00:00 to 24:00:00
Monday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Tuesday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Wednesday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Thursday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Friday	00:00:00 to 18:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Brent Crude Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)
2026-04-03

Early Closes

The following table shows the early closes for the Brent Crude Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/New York)**
2013-01-21	13:30:00

2013-02-18	13:30:00
2013-05-27	13:30:00
2013-07-04	13:30:00
2013-09-02	13:30:00
2013-11-28	13:30:00
2013-11-29	14:00:00
2013-12-24	14:00:00
2013-12-31	15:00:00
2014-01-20	13:30:00
2014-02-17	13:30:00
2014-05-26	13:30:00
2014-07-04	13:30:00
2014-09-01	13:30:00
2014-11-27	13:30:00
2014-11-28	14:00:00
2014-12-24	14:00:00
2014-12-31	15:00:00
2015-01-19	13:30:00
2015-02-16	13:30:00
2015-06-03	13:30:00
2015-07-03	13:30:00
2015-09-07	13:30:00
2015-11-26	13:30:00
2015-11-27	14:00:00
2015-12-24	14:00:00
2015-12-31	15:00:00
2016-01-18	13:30:00
2016-02-15	13:30:00

2016-05-30	13:30:00
2016-07-04	13:30:00
2016-09-05	13:30:00
2016-11-26	13:30:00
2016-11-27	14:00:00
2016-12-24	14:00:00
2016-12-30	15:00:00
2017-09-04	13:30:00
2017-11-23	13:30:00
2017-11-24	14:00:00
2017-12-22	17:00:00
2017-12-29	17:00:00
2018-07-04	13:30:00
2018-09-03	13:30:00
2018-11-22	13:30:00
2018-11-23	14:00:00
2018-12-24	17:00:00
2018-12-31	17:00:00
2019-05-27	13:30:00
2019-07-04	13:30:00
2019-09-02	13:30:00
2019-11-28	13:30:00
2019-11-29	14:00:00
2019-12-24	14:00:00
2019-12-31	15:00:00
2020-05-25	13:30:00
2020-07-03	13:30:00
2020-09-07	13:30:00

2020-11-26	13:30:00
2020-11-27	14:00:00
2020-12-24	14:00:00
2020-12-31	15:00:00

Late Opens

There are no days with late opens.

Time Zone

The Brent Crude Futures contract in the ICE Future market trades in the `America/New York` time zone.

ICE

CC

Introduction

This page shows the trading hours, holidays, and time zone of the Cocoa Futures contract in the ICE Future market.

Historical data for backtesting is unavailable for ICE. In live trading, LEAN sources this data from your brokerage or a third-party data provider .

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Cocoa Futures contract in the ICE Future market:

Weekday	**Time (America/New York)**
Monday	04:45:00 to 13:30:00
Tuesday	04:45:00 to 13:30:00
Wednesday	04:45:00 to 13:30:00
Thursday	04:45:00 to 13:30:00
Friday	04:45:00 to 13:30:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Cocoa Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)				
2025-02-17	2025-05-26	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2026-01-19	2026-02-16	2026-04-03	2026-05-25
2026-06-19	2026-07-03	2026-09-07	2026-11-26	

Early Closes

The following table shows the early closes for the Cocoa Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2008-11-26	14:15:00
2008-11-28	13:00:00
2008-12-24	13:00:00
2008-12-26	14:15:00
2009-11-27	13:00:00
2009-12-24	13:00:00
2010-11-26	13:00:00
2011-11-25	13:00:00
2012-11-23	13:00:00
2012-12-24	13:00:00
2013-11-29	13:00:00
2013-12-24	13:00:00
2014-11-28	13:00:00
2014-12-24	13:00:00
2015-11-27	13:00:00
2015-12-24	13:05:00
2016-11-25	13:00:00
2017-12-24	13:05:00
2019-12-24	13:05:00

Late Opens

The following table shows the late opens for the Cocoa Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2009-12-28	07:30:00
2010-12-27	07:30:00
2010-12-28	07:30:00
2011-12-27	07:30:00
2012-04-09	07:30:00
2012-12-26	07:30:00
2013-01-02	07:30:00
2013-04-01	07:30:00
2013-12-26	08:00:00
2014-01-02	08:00:00
2014-04-21	07:30:00
2014-12-26	08:00:00
2015-01-02	08:00:00
2015-04-06	07:30:00
2015-12-28	08:00:00
2016-12-27	08:00:00
2017-04-17	07:30:00
2017-12-26	08:00:00
2018-04-02	07:30:00
2018-12-26	08:00:00
2019-04-22	07:30:00
2019-12-26	08:00:00

Time Zone

The Cocoa Futures contract in the ICE Future market trades in the `America/New York` time zone.

ICE

CT

Introduction

This page shows the trading hours, holidays, and time zone of the Cotton No. 2 Futures contract in the ICE Future market.

Historical data for backtesting is unavailable for ICE. In live trading, LEAN sources this data from your brokerage or a third-party data provider .

Pre-market Hours

The following table shows the pre-market hours for the Cotton No. 2 Futures contract in the ICE Future market:

Weekday	**Time (America/New York)**
Sunday	19:30:00 to 21:00:00
Monday	19:30:00 to 21:00:00
Tuesday	19:30:00 to 21:00:00
Wednesday	19:30:00 to 21:00:00
Thursday	19:30:00 to 21:00:00

Regular Trading Hours

The following table shows the regular trading hours for the Cotton No. 2 Futures contract in the ICE Future market:

Weekday	**Time (America/New York)**
Sunday	21:00:00 to 24:00:00
Monday	00:00:00 to 14:20:00, 21:00:00 to 24:00:00
Tuesday	00:00:00 to 14:20:00, 21:00:00 to 24:00:00
Wednesday	00:00:00 to 14:20:00, 21:00:00 to 24:00:00
Thursday	00:00:00 to 14:20:00, 21:00:00 to 24:00:00
Friday	00:00:00 to 14:20:00

Post-market Hours

The following table shows the post-market hours for the Cotton No. 2 Futures contract in the ICE Future market:

Weekday	Time (America/New York)
Monday	14:50:00 to 18:00:00
Tuesday	14:50:00 to 18:00:00
Wednesday	14:50:00 to 18:00:00
Thursday	14:50:00 to 18:00:00
Friday	14:50:00 to 18:00:00

Holidays

The following table shows the dates of holidays for the Cotton No. 2 Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)				
2025-02-17	2025-05-26	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2026-01-19	2026-02-16	2026-04-03	2026-05-25
2026-06-19	2026-07-03	2026-09-07	2026-11-26	

Early Closes

The following table shows the early closes for the Cotton No. 2 Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2008-11-26	14:15:00
2008-11-28	13:00:00
2008-12-24	13:00:00
2008-12-26	14:15:00
2009-11-27	13:00:00
2009-12-24	13:00:00
2010-11-26	13:00:00
2011-11-25	13:00:00
2012-11-23	13:00:00
2012-12-24	13:00:00
2013-11-29	13:00:00
2013-12-24	13:00:00
2014-11-28	13:00:00
2014-12-24	13:00:00
2015-11-27	13:00:00
2015-12-24	13:05:00
2016-11-25	13:00:00
2017-11-23	13:30:00
2017-11-24	13:30:00
2017-12-24	13:05:00
2019-11-29	13:30:00
2019-12-24	13:05:00

Late Opens

The following table shows the late opens for the Cotton No. 2 Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2009-12-28	07:30:00
2010-12-27	07:30:00
2010-12-28	07:30:00
2011-12-27	07:30:00
2012-12-26	07:30:00
2013-01-02	07:30:00
2013-07-05	08:00:00
2013-11-29	08:00:00
2013-12-26	08:00:00
2014-01-02	08:00:00
2014-07-07	08:00:00
2014-11-28	08:00:00
2014-12-26	08:00:00
2015-01-02	08:00:00
2015-11-27	08:00:00
2015-12-28	08:00:00
2016-11-25	08:00:00
2017-11-24	08:00:00
2017-12-26	08:00:00
2018-11-23	08:00:00
2018-12-26	08:00:00
2019-11-29	08:00:00
2019-12-26	08:00:00

Time Zone

The Cotton No. 2 Futures contract in the ICE Future market trades in the `America/New York` time zone.

ICE

DX

Introduction

This page shows the trading hours, holidays, and time zone of the US Dollar Index Futures contract in the ICE Future market.

Historical data for backtesting is unavailable for ICE. In live trading, LEAN sources this data from your brokerage or a third-party data provider .

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US Dollar Index Futures contract in the ICE Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 20:00:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 20:00:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 20:00:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 20:00:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the ICE Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23

2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03

2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19
2024-07-04	2024-09-02	2024-11-28	2024-12-25	2025-01-01
2025-01-20	2025-04-18	2025-12-25	2026-01-01	2026-12-25
2027-01-01				

Early Closes

The following table shows the early closes for the US Dollar Index Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/New York)**
2007-12-24	13:15:00
2007-12-31	16:15:00
2008-01-21	13:00:00
2008-02-18	13:00:00
2008-03-20	16:15:00

2008-07-03	16:15:00
2008-07-04	13:00:00
2008-08-29	16:15:00
2008-09-01	13:00:00
2008-11-26	17:00:00
2008-11-27	13:00:00
2008-11-28	13:15:00
2008-12-24	13:15:00
2008-12-26	17:00:00
2008-12-31	17:00:00
2009-01-02	17:00:00
2009-01-16	16:15:00
2009-01-19	13:00:00
2009-02-13	16:15:00
2009-02-16	13:00:00
2009-04-09	16:15:00
2009-05-22	16:15:00
2009-05-25	13:00:00
2009-07-02	16:15:00
2009-07-03	13:00:00
2009-09-04	16:15:00
2009-09-07	13:00:00
2009-10-09	16:15:00
2009-11-25	17:00:00
2009-11-26	13:00:00
2009-11-27	13:15:00
2009-12-24	13:15:00
2009-12-31	17:00:00

2010-01-15	16:15:00
2010-01-18	13:00:00
2010-02-12	16:15:00
2010-02-15	13:00:00
2010-04-02	11:15:00
2010-05-28	16:15:00
2010-05-31	13:00:00
2010-07-02	16:15:00
2010-07-05	13:00:00
2010-09-03	16:15:00
2010-09-06	13:00:00
2010-10-08	16:15:00
2010-11-25	13:00:00
2010-11-26	13:15:00
2010-12-23	17:00:00
2010-12-31	13:15:00
2011-01-14	16:15:00
2011-01-17	13:00:00
2011-02-18	16:15:00
2011-02-21	13:00:00
2011-05-27	16:15:00
2011-05-30	13:00:00
2011-07-01	16:15:00
2011-07-04	13:00:00
2011-09-02	16:15:00
2011-09-05	13:00:00
2011-10-07	16:15:00
2011-11-24	13:00:00

2011-11-25	13:15:00
2012-02-17	16:15:00
2012-02-20	13:00:00
2012-04-06	11:15:00
2012-05-25	16:15:00
2012-05-28	13:00:00
2012-07-04	13:00:00
2012-08-31	16:15:00
2012-09-03	13:00:00
2012-10-05	16:15:00
2012-11-22	13:00:00
2012-11-23	13:15:00
2012-12-24	13:15:00
2013-02-15	16:15:00
2013-02-18	13:00:00
2013-05-24	16:15:00
2013-05-27	13:00:00
2013-07-04	13:00:00
2013-08-30	16:15:00
2013-09-02	13:00:00
2013-11-28	13:00:00
2013-11-29	13:15:00
2013-12-24	13:15:00
2014-01-17	16:15:00
2014-01-20	13:00:00
2014-02-14	16:15:00
2014-02-17	13:00:00
2014-05-26	13:00:00

2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:15:00
2014-12-24	13:15:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-04-03	11:15:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:15:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:15:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:15:00
2018-01-15	13:00:00

2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:15:00
2018-12-24	13:15:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:15:00
2019-12-24	13:15:00
2020-01-20	13:00:00

Late Opens

The following table shows the late opens for the US Dollar Index Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2008-12-26	06:00:00
2009-01-02	06:00:00
2011-12-27	06:00:00
2012-01-03	06:00:00
2012-12-26	06:00:00
2013-01-02	06:00:00
2013-12-26	06:00:00
2014-01-02	06:00:00

Time Zone

The US Dollar Index Futures contract in the ICE Future market trades in the `America/New York` time zone.

ICE

G

Introduction

This page shows the trading hours, holidays, and time zone of the Low Sulfur Gasoil contract in the ICE Future market.

Historical data for backtesting is unavailable for ICE. In live trading, LEAN sources this data from your brokerage or a third-party data provider .

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Low Sulfur Gasoil contract in the ICE Future market:

Weekday	**Time (America/New York)**
Sunday	20:00:00 to 24:00:00
Monday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Tuesday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Wednesday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Thursday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Friday	00:00:00 to 18:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Low Sulfur Gasoil contract in the ICE Future market:

Date (*yyyy-mm-dd*)
2026-04-03

Early Closes

The following table shows the early closes for the Low Sulfur Gasoil contract in the ICE Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/New York)**
2013-01-21	13:30:00

2013-02-18	13:30:00
2013-05-27	13:30:00
2013-07-04	13:30:00
2013-09-02	13:30:00
2013-11-28	13:30:00
2013-11-29	14:00:00
2013-12-24	14:00:00
2013-12-31	15:00:00
2014-01-20	13:30:00
2014-02-17	13:30:00
2014-05-26	13:30:00
2014-07-04	13:30:00
2014-09-01	13:30:00
2014-11-27	13:30:00
2014-11-28	14:00:00
2014-12-24	14:00:00
2014-12-31	15:00:00
2015-01-19	13:30:00
2015-02-16	13:30:00
2015-06-03	13:30:00
2015-07-03	13:30:00
2015-09-07	13:30:00
2015-11-26	13:30:00
2015-11-27	14:00:00
2015-12-24	14:00:00
2015-12-31	15:00:00
2016-01-18	13:30:00
2016-02-15	13:30:00

2016-05-30	13:30:00
2016-07-04	13:30:00
2016-09-05	13:30:00
2016-11-26	13:30:00
2016-11-27	14:00:00
2016-12-24	14:00:00
2016-12-30	15:00:00
2017-09-04	13:30:00
2017-11-23	13:30:00
2017-11-24	14:00:00
2017-12-22	17:00:00
2017-12-29	17:00:00
2018-07-04	13:30:00
2018-09-03	13:30:00
2018-11-22	13:30:00
2018-11-23	14:00:00
2018-12-24	17:00:00
2018-12-31	17:00:00
2019-05-27	13:30:00
2019-07-04	13:30:00
2019-09-02	13:30:00
2019-11-28	13:30:00
2019-11-29	14:00:00
2019-12-24	14:00:00
2019-12-31	15:00:00
2020-05-25	13:30:00
2020-07-03	13:30:00
2020-09-07	13:30:00

2020-11-26	13:30:00
2020-11-27	14:00:00
2020-12-24	14:00:00
2020-12-31	15:00:00

Late Opens

There are no days with late opens.

Time Zone

The Low Sulfur Gasoil contract in the ICE Future market trades in the `America/New York` time zone.

ICE

KC

Introduction

This page shows the trading hours, holidays, and time zone of the Coffee C Arabica Futures contract in the ICE Future market.

Historical data for backtesting is unavailable for ICE. In live trading, LEAN sources this data from your brokerage or a third-party data provider .

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Coffee C Arabica Futures contract in the ICE Future market:

Weekday	**Time (America/New York)**
Monday	04:15:00 to 13:30:00
Tuesday	04:15:00 to 13:30:00
Wednesday	04:15:00 to 13:30:00
Thursday	04:15:00 to 13:30:00
Friday	04:15:00 to 13:30:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Coffee C Arabica Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)				
2025-02-17	2025-05-26	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2026-01-19	2026-02-16	2026-04-03	2026-05-25
2026-06-19	2026-07-03	2026-09-07	2026-11-26	

Early Closes

The following table shows the early closes for the Coffee C Arabica Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2008-11-26	14:15:00
2008-11-28	13:00:00
2008-12-24	13:00:00
2008-12-26	14:15:00
2009-11-27	13:00:00
2009-12-24	13:00:00
2010-11-26	13:00:00
2011-11-25	13:00:00
2012-11-23	13:00:00
2012-12-24	13:00:00
2013-11-29	13:00:00
2013-12-24	13:00:00
2014-11-28	13:00:00
2014-12-24	13:00:00
2015-11-27	13:00:00
2015-12-24	13:05:00
2016-11-25	13:00:00
2017-12-24	13:05:00
2019-12-24	13:05:00

Late Opens

The following table shows the late opens for the Coffee C Arabica Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2009-12-28	07:30:00
2010-12-27	07:30:00
2010-12-28	07:30:00
2011-12-27	07:30:00
2012-04-09	07:30:00
2012-12-26	07:30:00
2013-01-02	07:30:00
2013-04-01	07:30:00
2013-12-26	08:00:00
2014-01-02	08:00:00
2014-04-21	07:30:00
2014-12-26	08:00:00
2015-01-02	08:00:00
2015-04-06	07:30:00
2015-12-28	08:00:00
2016-12-27	08:00:00
2017-04-17	07:30:00
2017-12-26	08:00:00
2018-04-02	07:30:00
2018-12-26	08:00:00
2019-04-22	07:30:00
2019-12-26	08:00:00

Time Zone

The Coffee C Arabica Futures contract in the ICE Future market trades in the `America/New York` time zone.

ICE

OJ

Introduction

This page shows the trading hours, holidays, and time zone of the Frozen Concentrated Orange Juice contract in the ICE Future market.

Historical data for backtesting is unavailable for ICE. In live trading, LEAN sources this data from your brokerage or a third-party data provider .

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Frozen Concentrated Orange Juice contract in the ICE Future market:

Weekday	**Time (America/New York)**
Monday	08:00:00 to 14:00:00
Tuesday	08:00:00 to 14:00:00
Wednesday	08:00:00 to 14:00:00
Thursday	08:00:00 to 14:00:00
Friday	08:00:00 to 14:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Frozen Concentrated Orange Juice contract in the ICE Future market:

Date (*yyyy-mm-dd*)				
2025-02-17	2025-05-26	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2026-01-19	2026-02-16	2026-04-03	2026-05-25
2026-06-19	2026-07-03	2026-09-07	2026-11-26	

Early Closes

The following table shows the early closes for the Frozen Concentrated Orange Juice contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2008-11-26	14:15:00
2008-11-28	13:00:00
2008-12-24	13:00:00
2008-12-26	14:15:00
2009-11-27	13:00:00
2009-12-24	13:00:00
2010-11-26	13:00:00
2011-11-25	13:00:00
2012-11-23	13:00:00
2012-12-24	13:00:00
2013-11-29	13:00:00
2013-12-24	13:00:00
2014-11-28	13:00:00
2014-12-24	13:00:00
2015-11-27	13:00:00
2015-12-24	13:05:00
2016-11-25	13:00:00
2017-11-23	13:30:00
2017-11-24	13:30:00
2017-12-24	13:05:00
2019-11-29	13:30:00
2019-12-24	13:05:00

Late Opens

The following table shows the late opens for the Frozen Concentrated Orange Juice contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2012-12-26	07:30:00

Time Zone

The Frozen Concentrated Orange Juice contract in the ICE Future market trades in the `America/New York` time zone.

ICE

SB

Introduction

This page shows the trading hours, holidays, and time zone of the Sugar No. 11 Futures contract in the ICE Future market.

Historical data for backtesting is unavailable for ICE. In live trading, LEAN sources this data from your brokerage or a third-party data provider .

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Sugar No. 11 Futures contract in the ICE Future market:

Weekday	**Time (America/New York)**
Monday	03:30:00 to 13:00:00
Tuesday	03:30:00 to 13:00:00
Wednesday	03:30:00 to 13:00:00
Thursday	03:30:00 to 13:00:00
Friday	03:30:00 to 13:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Sugar No. 11 Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)				
2025-02-17	2025-05-26	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2026-01-19	2026-02-16	2026-04-03	2026-05-25
2026-06-19	2026-07-03	2026-09-07	2026-11-26	

Early Closes

The following table shows the early closes for the Sugar No. 11 Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2008-11-26	14:15:00
2008-11-28	13:00:00
2008-12-24	13:00:00
2008-12-26	14:15:00
2009-11-27	13:00:00
2009-12-24	13:00:00
2010-11-26	13:00:00
2011-11-25	13:00:00
2012-11-23	13:00:00
2012-12-24	13:00:00
2013-11-29	13:00:00
2013-12-24	13:00:00

Late Opens

The following table shows the late opens for the Sugar No. 11 Futures contract in the ICE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2009-12-28	07:30:00
2010-12-27	07:30:00
2010-12-28	07:30:00
2011-12-27	07:30:00
2012-04-09	07:30:00
2012-12-26	07:30:00
2013-01-02	07:30:00
2013-04-01	07:30:00
2013-12-26	08:00:00
2014-01-02	08:00:00
2014-04-21	07:30:00
2014-12-26	08:00:00
2015-01-02	08:00:00
2015-04-06	07:30:00
2015-12-28	08:00:00
2016-12-27	08:00:00
2017-04-17	07:30:00
2017-12-26	08:00:00
2018-04-02	07:30:00
2018-12-26	08:00:00
2019-04-22	07:30:00
2019-12-26	08:00:00

Time Zone

The Sugar No. 11 Futures contract in the ICE Future market trades in the `America/New York` time zone.

Market Hours

INDIA

Introduction

This page shows the trading hours, holidays, and time zone of the INDIA Future market.

Historical data for backtesting is unavailable for INDIA. In live trading, LEAN sources this data from your brokerage or a third-party data provider .

Pre-market Hours

The following table shows the pre-market hours for the INDIA Future market:

Weekday	**Time (Asia/Kolkata)**
Monday	09:00:00 to 09:15:00
Tuesday	09:00:00 to 09:15:00
Wednesday	09:00:00 to 09:15:00
Thursday	09:00:00 to 09:15:00
Friday	09:00:00 to 09:15:00

Regular Trading Hours

The following table shows the regular trading hours for the INDIA Future market:

Weekday	**Time (Asia/Kolkata)**
Monday	09:15:00 to 15:30:00
Tuesday	09:15:00 to 15:30:00
Wednesday	09:15:00 to 15:30:00
Thursday	09:15:00 to 15:30:00
Friday	09:15:00 to 15:30:00

Post-market Hours

The following table shows the post-market hours for the INDIA Future market:

Weekday	Time (Asia/Kolkata)
Monday	15:40:00 to 16:00:00
Tuesday	15:40:00 to 16:00:00
Wednesday	15:40:00 to 16:00:00
Thursday	15:40:00 to 16:00:00
Friday	15:40:00 to 16:00:00

Holidays

The following table shows the dates of holidays for the INDIA Future market:

Date (*yyyy-mm-dd*)				
2004-01-26	2004-04-14	2005-01-26	2005-04-14	2005-08-15
2006-01-26	2006-04-14	2006-05-01	2006-08-15	2006-10-02
2006-12-25	2007-01-26	2007-05-01	2007-08-15	2007-10-02
2007-12-25	2008-04-14	2008-05-01	2008-08-15	2008-10-02
2008-12-25	2009-01-26	2009-04-14	2009-05-01	2009-10-02
2009-12-25	2010-01-26	2010-04-14	2011-01-26	2011-03-02
2011-04-12	2011-04-14	2011-04-22	2011-08-15	2011-08-31
2011-09-01	2011-10-06	2011-10-26	2011-10-27	2011-11-07
2011-11-10	2011-12-06	2012-01-26	2012-02-20	2012-03-08
2012-04-05	2012-04-06	2012-05-01	2012-08-15	2012-08-20
2012-09-19	2012-10-02	2012-10-24	2012-11-14	2012-11-28
2012-12-25	2013-03-27	2013-03-29	2013-04-19	2013-04-24
2013-05-01	2013-08-09	2013-08-15	2013-09-09	2013-10-02
2013-10-16	2013-11-04	2013-11-15	2013-12-25	2014-02-27
2014-03-17	2014-04-08	2014-04-14	2014-04-18	2014-04-24
2014-05-01	2014-07-29	2014-08-15	2014-08-29	2014-10-02
2014-10-03	2014-10-06	2014-10-15	2014-10-24	2014-11-04
2014-11-06	2014-12-25	2015-01-26	2015-02-17	2015-03-06
2015-04-02	2015-04-03	2015-04-14	2015-05-01	2015-09-17

2015-09-25	2015-10-02	2015-10-22	2015-11-12	2015-11-25
2015-12-25	2016-01-26	2016-03-07	2016-03-24	2016-03-25
2016-04-14	2016-04-15	2016-04-19	2016-07-06	2016-08-15
2016-09-05	2016-09-13	2016-10-11	2016-10-12	2016-10-31
2016-11-14	2017-01-26	2017-02-24	2017-03-13	2017-04-04
2017-04-14	2017-05-01	2017-06-26	2017-08-15	2017-08-25
2017-10-02	2017-10-20	2017-12-25	2018-01-26	2018-02-13
2018-03-02	2018-03-29	2018-03-30	2018-05-01	2018-08-15
2018-08-22	2018-09-13	2018-09-20	2018-10-02	2018-10-18
2018-11-08	2018-11-23	2018-12-25	2019-03-04	2019-03-21
2019-04-17	2019-04-19	2019-04-29	2019-05-01	2019-06-05
2019-08-12	2019-08-15	2019-09-02	2019-09-10	2019-10-02
2019-10-08	2019-10-21	2019-10-28	2019-11-12	2019-12-25
2020-02-21	2020-03-10	2020-04-02	2020-04-06	2020-04-10
2020-04-14	2020-05-01	2020-05-25	2020-10-02	2020-11-16
2020-11-30	2020-12-25	2021-01-26	2021-03-11	2021-03-29
2021-04-02	2021-04-14	2021-04-21	2021-05-13	2021-07-21
2021-08-19	2021-09-10	2021-10-15	2021-11-05	2021-11-19
2022-01-26	2022-04-14	2022-08-15	2023-01-26	2023-04-14
2023-05-01	2023-08-15	2023-10-02	2023-12-25	2024-01-26
2024-05-01	2024-08-15	2024-10-02	2024-12-25	2025-02-26
2025-03-14	2025-03-31	2025-04-10	2025-04-14	2025-04-18
2025-05-01	2025-08-15	2025-10-02	2025-10-22	2025-12-25
2026-01-26	2026-04-14	2026-05-01	2026-10-02	2026-12-25

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The INDIA Future market trades in the `Asia/Kolkata` time zone.

Market Hours

NYMEX

Introduction

This page shows the trading hours, holidays, and time zone of the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04

2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19

2023-07-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-20	2025-04-18

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The NYMEX Future market trades in the following time zones:

- `America/Chicago`
- `America/New York`

Assets With Other Hours

The following table shows the contracts that have different trading periods than the overall NYMEX Future market:

Symbol	Name
1S	Propane Non-LDH Mont Belvieu (OPIS) BALMO Futures
22	Argus Propane Far East Index BALMO Futures
A0D	Mini European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures
A0F	Mini Singapore Fuel Oil 180 cst (Platts) Futures
A1L	Gulf Coast ULSD (Platts) Up-Down BALMO Futures
A1M	Gulf Coast Jet (Platts) Up-Down BALMO Futures
A1R	Propane Non-LDH Mont Belvieu (OPIS) Futures
A32	European Propane CIF ARA (Argus) BALMO Futures
A3G	Premium Unleaded Gasoline 10 ppm FOB MED (Platts) Futures
A7E	Argus Propane Far East Index Futures
A7I	Gasoline Euro-bob Oxy NWE Barges (Argus) Crack Spread BALMO Futures
A7Q	Mont Belvieu Natural Gasoline (OPIS) Futures
A8J	Mont Belvieu Normal Butane (OPIS) BALMO Futures
A8K	Conway Propane (OPIS) Futures

A8O	Mont Belvieu LDH Propane (OPIS) BALMO Futures
A91	Argus Propane Far East Index vs. European Propane CIF ARA (Argus) Futures
A9N	Argus Propane (Saudi Aramco) Futures
AA6	Group Three ULSD (Platts) vs. NY Harbor ULSD Futures
AA8	Group Three Sub-octane Gasoline (Platts) vs. RBOB Futures
ABS	Singapore Fuel Oil 180 cst (Platts) BALMO Futures
ABT	Singapore Fuel Oil 380 cst (Platts) BALMO Futures
AC0	Mont Belvieu Ethane (OPIS) Futures
AD0	Mont Belvieu Normal Butane (OPIS) Futures
ADB	Brent Crude Oil vs. Dubai Crude Oil (Platts) Futures
AE5	Argus LLS vs. WTI (Argus) Trade Month Futures
AGA	Singapore Gasoil (Platts) vs. Low Sulphur Gasoil Futures
AJL	Los Angeles CARBOB Gasoline (OPIS) vs. RBOB Gasoline Futures
AJS	Los Angeles Jet (OPIS) vs. NY Harbor ULSD Futures
AKL	Los Angeles CARB Diesel (OPIS) vs. NY Harbor ULSD Futures
AKZ	European Naphtha (Platts) BALMO Futures
APS	European Propane CIF ARA (Argus) Futures
AR0	Mont Belvieu Natural Gasoline (OPIS) BALMO Futures
ARE	RBOB Gasoline Crack Spread Futures
AVZ	Gulf Coast HSFO (Platts) BALMO Futures
AYV	Mars (Argus) vs. WTI Trade Month Futures
AYX	Mars (Argus) vs. WTI Financial Futures
AZ1	Ethanol T2 FOB Rdam Including Duty (Platts) Futures
B0	Mont Belvieu LDH Propane (OPIS) Futures
B7H	Gasoline Euro-bob Oxy NWE Barges (Argus) Futures
BK	WTI-Brent Financial Futures
BOO	3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread (1000mt) Futures

BR7	Gasoline Euro-bob Oxy NWE Barges (Argus) BALMO Futures
BZ	Brent Last Day Financial Futures
CL	Crude Oil Futures
CRB	Gulf Coast CBOB Gasoline A2 (Platts) vs. RBOB Gasoline Futures
CSW	CSW
CSX	WTI Financial Futures
CU	Chicago Ethanol (Platts) Futures
D1N	Singapore Mogas 92 Unleaded (Platts) Brent Crack Spread Futures
DCB	Dubai Crude Oil (Platts) Financial Futures
E6	Japan C&F Naphtha (Platts) BALMO Futures
EN	European Naphtha (Platts) Crack Spread Futures
EPN	European Propane CIF ARA (Argus) vs. Naphtha Cargoes CIF NWE (Platts) Futures
EVC	Singapore Fuel Oil 380 cst (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures
EWG	East-West Gasoline Spread (Platts-Argus) Futures
EWN	East-West Naphtha: Japan C&F vs. Cargoes CIF NWE Spread (Platts) Futures
EXR	RBOB Gasoline vs. Euro-bob Oxy NWE Barges (Argus) (350000 gallons) Futures
FO	3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread Futures
FRC	Freight Route TC14 (Baltic) Futures
FSS	1% Fuel Oil Cargoes FOB NWE (Platts) vs. 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures
GCU	Gulf Coast HSFO (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures
HCL	WTI Houston Crude Oil Futures
HH	Natural Gas (Henry Hub) Last-day Financial Futures
HO	NY Harbor ULSD Futures

HP	Natural Gas (Henry Hub) Penultimate Financial Futures
HRC	U.S. Midwest Domestic Hot-Rolled Coil Steel (CRU) Index Futures
HTT	WTI Houston (Argus) vs. WTI Trade Month Futures
M1B	Micro Gasoil 0.1% Barges FOB ARA (Platts) Futures
M35	Micro European 3.5% Fuel Oil Cargoes FOB Med (Platts) Futures
M5F	Micro Coal (API 5) fob Newcastle (Argus/McCloskey) Futures
MAF	Micro Singapore Fuel Oil 380CST (Platts) Futures
MCL	Micro WTI Crude Oil Futures
MEF	Micro European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures
NG	Henry Hub Natural Gas Futures
PA	Palladium Futures
PAM	Micro Palladium Futures
PL	Platinum Futures
R5O	Micro European FOB Rdam Marine Fuel 0.5% Barges (Platts) Futures
RB	RBOB Gasoline Futures
S5O	Micro Singapore FOB Marine Fuel 0.5% (Platts) Futures
YO	No. 11 Sugar Futures

NYMEX

1S

Introduction

This page shows the trading hours, holidays, and time zone of the Propane Non-LDH Mont Belvieu (OPIS) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Propane Non-LDH Mont Belvieu (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Propane Non-LDH Mont Belvieu (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Propane Non-LDH Mont Belvieu (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Propane Non-LDH Mont Belvieu (OPIS) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Propane Non-LDH Mont Belvieu (OPIS) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Propane Non-LDH Mont Belvieu (OPIS) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Propane Non-LDH Mont Belvieu (OPIS) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

22

Introduction

This page shows the trading hours, holidays, and time zone of the Argus Propane Far East Index BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Argus Propane Far East Index BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Argus Propane Far East Index BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Argus Propane Far East Index BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Argus Propane Far East Index BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Argus Propane Far East Index BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Argus Propane Far East Index BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Argus Propane Far East Index BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A0D

Introduction

This page shows the trading hours, holidays, and time zone of the Mini European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mini European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mini European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mini European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Mini European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Mini European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Mini European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Mini European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A0F

Introduction

This page shows the trading hours, holidays, and time zone of the Mini Singapore Fuel Oil 180 cst (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mini Singapore Fuel Oil 180 cst (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mini Singapore Fuel Oil 180 cst (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mini Singapore Fuel Oil 180 cst (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Mini Singapore Fuel Oil 180 cst (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Mini Singapore Fuel Oil 180 cst (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Mini Singapore Fuel Oil 180 cst (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Mini Singapore Fuel Oil 180 cst (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A1L

Introduction

This page shows the trading hours, holidays, and time zone of the Gulf Coast ULSD (Platts) Up-Down BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Gulf Coast ULSD (Platts) Up-Down BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Gulf Coast ULSD (Platts) Up-Down BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Gulf Coast ULSD (Platts) Up-Down BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Gulf Coast ULSD (Platts) Up-Down BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Gulf Coast ULSD (Platts) Up-Down BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Gulf Coast ULSD (Platts) Up-Down BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Gulf Coast ULSD (Platts) Up-Down BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A1M

Introduction

This page shows the trading hours, holidays, and time zone of the Gulf Coast Jet (Platts) Up-Down BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Gulf Coast Jet (Platts) Up-Down BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Gulf Coast Jet (Platts) Up-Down BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Gulf Coast Jet (Platts) Up-Down BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Gulf Coast Jet (Platts) Up-Down BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Gulf Coast Jet (Platts) Up-Down BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Gulf Coast Jet (Platts) Up-Down BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Gulf Coast Jet (Platts) Up-Down BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A1R

Introduction

This page shows the trading hours, holidays, and time zone of the Propane Non-LDH Mont Belvieu (OPIS) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Propane Non-LDH Mont Belvieu (OPIS) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Propane Non-LDH Mont Belvieu (OPIS) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Propane Non-LDH Mont Belvieu (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Propane Non-LDH Mont Belvieu (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Propane Non-LDH Mont Belvieu (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Propane Non-LDH Mont Belvieu (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Propane Non-LDH Mont Belvieu (OPIS) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A32

Introduction

This page shows the trading hours, holidays, and time zone of the European Propane CIF ARA (Argus) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the European Propane CIF ARA (Argus) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the European Propane CIF ARA (Argus) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the European Propane CIF ARA (Argus) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the European Propane CIF ARA (Argus) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the European Propane CIF ARA (Argus) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the European Propane CIF ARA (Argus) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The European Propane CIF ARA (Argus) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A3G

Introduction

This page shows the trading hours, holidays, and time zone of the Premium Unleaded Gasoline 10 ppm FOB MED (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Premium Unleaded Gasoline 10 ppm FOB MED (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Premium Unleaded Gasoline 10 ppm FOB MED (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Premium Unleaded Gasoline 10 ppm FOB MED (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Premium Unleaded Gasoline 10 ppm FOB MED (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Premium Unleaded Gasoline 10 ppm FOB MED (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Premium Unleaded Gasoline 10 ppm FOB MED (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Premium Unleaded Gasoline 10 ppm FOB MED (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A7E

Introduction

This page shows the trading hours, holidays, and time zone of the Argus Propane Far East Index Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Argus Propane Far East Index Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Argus Propane Far East Index Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Argus Propane Far East Index Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Argus Propane Far East Index Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Argus Propane Far East Index Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Argus Propane Far East Index Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00

2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00

2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00

2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00

2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Argus Propane Far East Index Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A7I

Introduction

This page shows the trading hours, holidays, and time zone of the Gasoline Euro-bob Oxy NWE Barges (Argus) Crack Spread BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Gasoline Euro-bob Oxy NWE Barges (Argus) Crack Spread BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Gasoline Euro-bob Oxy NWE Barges (Argus) Crack Spread BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Gasoline Euro-bob Oxy NWE Barges (Argus) Crack Spread BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Gasoline Euro-bob Oxy NWE Barges (Argus) Crack Spread BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Gasoline Euro-bob Oxy NWE Barges (Argus) Crack Spread BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Gasoline Euro-bob Oxy NWE Barges (Argus) Crack Spread BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Gasoline Euro-bob Oxy NWE Barges (Argus) Crack Spread BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A7Q

Introduction

This page shows the trading hours, holidays, and time zone of the Mont Belvieu Natural Gasoline (OPIS) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mont Belvieu Natural Gasoline (OPIS) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mont Belvieu Natural Gasoline (OPIS) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mont Belvieu Natural Gasoline (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Mont Belvieu Natural Gasoline (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Mont Belvieu Natural Gasoline (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Mont Belvieu Natural Gasoline (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Mont Belvieu Natural Gasoline (OPIS) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A8J

Introduction

This page shows the trading hours, holidays, and time zone of the Mont Belvieu Normal Butane (OPIS) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mont Belvieu Normal Butane (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mont Belvieu Normal Butane (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mont Belvieu Normal Butane (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Mont Belvieu Normal Butane (OPIS) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Mont Belvieu Normal Butane (OPIS) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Mont Belvieu Normal Butane (OPIS) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Mont Belvieu Normal Butane (OPIS) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A8K

Introduction

This page shows the trading hours, holidays, and time zone of the Conway Propane (OPIS) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Conway Propane (OPIS) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Conway Propane (OPIS) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Conway Propane (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Conway Propane (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Conway Propane (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00

2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00

2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00

2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00

2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00
2024-02-19	14:30:00

2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00

2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Conway Propane (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00

2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Conway Propane (OPIS) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A8O

Introduction

This page shows the trading hours, holidays, and time zone of the Mont Belvieu LDH Propane (OPIS) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mont Belvieu LDH Propane (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mont Belvieu LDH Propane (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mont Belvieu LDH Propane (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Mont Belvieu LDH Propane (OPIS) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Mont Belvieu LDH Propane (OPIS) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Mont Belvieu LDH Propane (OPIS) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Mont Belvieu LDH Propane (OPIS) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A91

Introduction

This page shows the trading hours, holidays, and time zone of the Argus Propane Far East Index vs. European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Argus Propane Far East Index vs. European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Argus Propane Far East Index vs. European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Argus Propane Far East Index vs. European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Argus Propane Far East Index vs. European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Argus Propane Far East Index vs. European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Argus Propane Far East Index vs. European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Argus Propane Far East Index vs. European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

A9N

Introduction

This page shows the trading hours, holidays, and time zone of the Argus Propane (Saudi Aramco) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Argus Propane (Saudi Aramco) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Argus Propane (Saudi Aramco) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Argus Propane (Saudi Aramco) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Argus Propane (Saudi Aramco) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Argus Propane (Saudi Aramco) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Argus Propane (Saudi Aramco) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00

2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00

2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00

2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00

2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Argus Propane (Saudi Aramco) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AA6

Introduction

This page shows the trading hours, holidays, and time zone of the Group Three ULSD (Platts) vs. NY Harbor ULSD Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Group Three ULSD (Platts) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Group Three ULSD (Platts) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Group Three ULSD (Platts) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Group Three ULSD (Platts) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Group Three ULSD (Platts) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Group Three ULSD (Platts) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Group Three ULSD (Platts) vs. NY Harbor ULSD Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AA8

Introduction

This page shows the trading hours, holidays, and time zone of the Group Three Sub-octane Gasoline (Platts) vs. RBOB Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Group Three Sub-octane Gasoline (Platts) vs. RBOB Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Group Three Sub-octane Gasoline (Platts) vs. RBOB Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Group Three Sub-octane Gasoline (Platts) vs. RBOB Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Group Three Sub-octane Gasoline (Platts) vs. RBOB Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Group Three Sub-octane Gasoline (Platts) vs. RBOB Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Group Three Sub-octane Gasoline (Platts) vs. RBOB Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Group Three Sub-octane Gasoline (Platts) vs. RBOB Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

ABS

Introduction

This page shows the trading hours, holidays, and time zone of the Singapore Fuel Oil 180 cst (Platts) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Singapore Fuel Oil 180 cst (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Singapore Fuel Oil 180 cst (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Singapore Fuel Oil 180 cst (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Singapore Fuel Oil 180 cst (Platts) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Singapore Fuel Oil 180 cst (Platts) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Singapore Fuel Oil 180 cst (Platts) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Singapore Fuel Oil 180 cst (Platts) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

ABT

Introduction

This page shows the trading hours, holidays, and time zone of the Singapore Fuel Oil 380 cst (Platts) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Singapore Fuel Oil 380 cst (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Singapore Fuel Oil 380 cst (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Singapore Fuel Oil 380 cst (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Singapore Fuel Oil 380 cst (Platts) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Singapore Fuel Oil 380 cst (Platts) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Singapore Fuel Oil 380 cst (Platts) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Singapore Fuel Oil 380 cst (Platts) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AC0

Introduction

This page shows the trading hours, holidays, and time zone of the Mont Belvieu Ethane (OPIS) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mont Belvieu Ethane (OPIS) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mont Belvieu Ethane (OPIS) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mont Belvieu Ethane (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Mont Belvieu Ethane (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Mont Belvieu Ethane (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Mont Belvieu Ethane (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00

2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00

2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00

2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00

2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Mont Belvieu Ethane (OPIS) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AD0

Introduction

This page shows the trading hours, holidays, and time zone of the Mont Belvieu Normal Butane (OPIS) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mont Belvieu Normal Butane (OPIS) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mont Belvieu Normal Butane (OPIS) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mont Belvieu Normal Butane (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Mont Belvieu Normal Butane (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Mont Belvieu Normal Butane (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Mont Belvieu Normal Butane (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Mont Belvieu Normal Butane (OPIS) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

ADB

Introduction

This page shows the trading hours, holidays, and time zone of the Brent Crude Oil vs. Dubai Crude Oil (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Brent Crude Oil vs. Dubai Crude Oil (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Brent Crude Oil vs. Dubai Crude Oil (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Brent Crude Oil vs. Dubai Crude Oil (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Brent Crude Oil vs. Dubai Crude Oil (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Brent Crude Oil vs. Dubai Crude Oil (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Brent Crude Oil vs. Dubai Crude Oil (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Brent Crude Oil vs. Dubai Crude Oil (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AE5

Introduction

This page shows the trading hours, holidays, and time zone of the Argus LLS vs. WTI (Argus) Trade Month Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Argus LLS vs. WTI (Argus) Trade Month Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Argus LLS vs. WTI (Argus) Trade Month Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Argus LLS vs. WTI (Argus) Trade Month Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Argus LLS vs. WTI (Argus) Trade Month Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Argus LLS vs. WTI (Argus) Trade Month Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Argus LLS vs. WTI (Argus) Trade Month Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Argus LLS vs. WTI (Argus) Trade Month Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AGA

Introduction

This page shows the trading hours, holidays, and time zone of the Singapore Gasoil (Platts) vs. Low Sulphur Gasoil Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Singapore Gasoil (Platts) vs. Low Sulphur Gasoil Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Singapore Gasoil (Platts) vs. Low Sulphur Gasoil Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Singapore Gasoil (Platts) vs. Low Sulphur Gasoil Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Singapore Gasoil (Platts) vs. Low Sulphur Gasoil Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Singapore Gasoil (Platts) vs. Low Sulphur Gasoil Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Singapore Gasoil (Platts) vs. Low Sulphur Gasoil Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Singapore Gasoil (Platts) vs. Low Sulphur Gasoil Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AJL

Introduction

This page shows the trading hours, holidays, and time zone of the Los Angeles CARBOB Gasoline (OPIS) vs. RBOB Gasoline Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Los Angeles CARBOB Gasoline (OPIS) vs. RBOB Gasoline Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Los Angeles CARBOB Gasoline (OPIS) vs. RBOB Gasoline Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Los Angeles CARBOB Gasoline (OPIS) vs. RBOB Gasoline Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Los Angeles CARBOB Gasoline (OPIS) vs. RBOB Gasoline Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Los Angeles CARBOB Gasoline (OPIS) vs. RBOB Gasoline Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Los Angeles CARBOB Gasoline (OPIS) vs. RBOB Gasoline Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Los Angeles CARBOB Gasoline (OPIS) vs. RBOB Gasoline Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AJS

Introduction

This page shows the trading hours, holidays, and time zone of the Los Angeles Jet (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Los Angeles Jet (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Los Angeles Jet (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Los Angeles Jet (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Los Angeles Jet (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Los Angeles Jet (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Los Angeles Jet (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Los Angeles Jet (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AKL

Introduction

This page shows the trading hours, holidays, and time zone of the Los Angeles CARB Diesel (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Los Angeles CARB Diesel (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Los Angeles CARB Diesel (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Los Angeles CARB Diesel (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Los Angeles CARB Diesel (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Los Angeles CARB Diesel (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Los Angeles CARB Diesel (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Los Angeles CARB Diesel (OPIS) vs. NY Harbor ULSD Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AKZ

Introduction

This page shows the trading hours, holidays, and time zone of the European Naphtha (Platts) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the European Naphtha (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the European Naphtha (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the European Naphtha (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the European Naphtha (Platts) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the European Naphtha (Platts) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the European Naphtha (Platts) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The European Naphtha (Platts) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

APS

Introduction

This page shows the trading hours, holidays, and time zone of the European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The European Propane CIF ARA (Argus) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AR0

Introduction

This page shows the trading hours, holidays, and time zone of the Mont Belvieu Natural Gasoline (OPIS) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mont Belvieu Natural Gasoline (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mont Belvieu Natural Gasoline (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mont Belvieu Natural Gasoline (OPIS) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Mont Belvieu Natural Gasoline (OPIS) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Mont Belvieu Natural Gasoline (OPIS) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Mont Belvieu Natural Gasoline (OPIS) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Mont Belvieu Natural Gasoline (OPIS) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

ARE

Introduction

This page shows the trading hours, holidays, and time zone of the RBOB Gasoline Crack Spread Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the RBOB Gasoline Crack Spread Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the RBOB Gasoline Crack Spread Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the RBOB Gasoline Crack Spread Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the RBOB Gasoline Crack Spread Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the RBOB Gasoline Crack Spread Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the RBOB Gasoline Crack Spread Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00

2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00

2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00

2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00

2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The RBOB Gasoline Crack Spread Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AVZ

Introduction

This page shows the trading hours, holidays, and time zone of the Gulf Coast HSFO (Platts) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Gulf Coast HSFO (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Gulf Coast HSFO (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Gulf Coast HSFO (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Gulf Coast HSFO (Platts) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Gulf Coast HSFO (Platts) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Gulf Coast HSFO (Platts) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00

2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00

2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00

2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00

2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Gulf Coast HSFO (Platts) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AYV

Introduction

This page shows the trading hours, holidays, and time zone of the Mars (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mars (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mars (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mars (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Mars (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Mars (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Mars (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Mars (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AYX

Introduction

This page shows the trading hours, holidays, and time zone of the Mars (Argus) vs. WTI Financial Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mars (Argus) vs. WTI Financial Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mars (Argus) vs. WTI Financial Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mars (Argus) vs. WTI Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Mars (Argus) vs. WTI Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Mars (Argus) vs. WTI Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Mars (Argus) vs. WTI Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00

2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00

2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00

2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00

2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Mars (Argus) vs. WTI Financial Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

AZ1

Introduction

This page shows the trading hours, holidays, and time zone of the Ethanol T2 FOB Rdam Including Duty (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Ethanol T2 FOB Rdam Including Duty (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Ethanol T2 FOB Rdam Including Duty (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Ethanol T2 FOB Rdam Including Duty (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Ethanol T2 FOB Rdam Including Duty (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Ethanol T2 FOB Rdam Including Duty (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Ethanol T2 FOB Rdam Including Duty (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Ethanol T2 FOB Rdam Including Duty (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

B0

Introduction

This page shows the trading hours, holidays, and time zone of the Mont Belvieu LDH Propane (OPIS) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Mont Belvieu LDH Propane (OPIS) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Mont Belvieu LDH Propane (OPIS) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Mont Belvieu LDH Propane (OPIS) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Mont Belvieu LDH Propane (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Mont Belvieu LDH Propane (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Mont Belvieu LDH Propane (OPIS) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Mont Belvieu LDH Propane (OPIS) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

B7H

Introduction

This page shows the trading hours, holidays, and time zone of the Gasoline Euro-bob Oxy NWE Barges (Argus) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Gasoline Euro-bob Oxy NWE Barges (Argus) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Gasoline Euro-bob Oxy NWE Barges (Argus) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Gasoline Euro-bob Oxy NWE Barges (Argus) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Gasoline Euro-bob Oxy NWE Barges (Argus) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Gasoline Euro-bob Oxy NWE Barges (Argus) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Gasoline Euro-bob Oxy NWE Barges (Argus) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Gasoline Euro-bob Oxy NWE Barges (Argus) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

BK

Introduction

This page shows the trading hours, holidays, and time zone of the WTI-Brent Financial Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the WTI-Brent Financial Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the WTI-Brent Financial Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the WTI-Brent Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the WTI-Brent Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the WTI-Brent Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00

2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00

2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00

2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00

2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00
2024-02-19	14:30:00

2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00

2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the WTI-Brent Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00

2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The WTI-Brent Financial Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

BOO

Introduction

This page shows the trading hours, holidays, and time zone of the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread (1000mt) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread (1000mt) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread (1000mt) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread (1000mt) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread (1000mt) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread (1000mt) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread (1000mt) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread (1000mt) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

BR7

Introduction

This page shows the trading hours, holidays, and time zone of the Gasoline Euro-bob Oxy NWE Barges (Argus) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Gasoline Euro-bob Oxy NWE Barges (Argus) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Gasoline Euro-bob Oxy NWE Barges (Argus) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Gasoline Euro-bob Oxy NWE Barges (Argus) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Gasoline Euro-bob Oxy NWE Barges (Argus) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Gasoline Euro-bob Oxy NWE Barges (Argus) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Gasoline Euro-bob Oxy NWE Barges (Argus) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Gasoline Euro-bob Oxy NWE Barges (Argus) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

BZ

Introduction

This page shows the trading hours, holidays, and time zone of the Brent Last Day Financial Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Brent Last Day Financial Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Brent Last Day Financial Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Brent Last Day Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Brent Last Day Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Brent Last Day Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00

2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00

2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00

2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00

2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00
2024-02-19	14:30:00

2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00

2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Brent Last Day Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00

2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Brent Last Day Financial Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

CL

Introduction

This page shows the trading hours, holidays, and time zone of the Crude Oil Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Crude Oil Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Crude Oil Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Crude Oil Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Crude Oil Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Crude Oil Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00

2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00

2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00

2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00

2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00
2024-02-19	14:30:00

2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00

2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Crude Oil Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00

2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Crude Oil Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

CRB

Introduction

This page shows the trading hours, holidays, and time zone of the Gulf Coast CBOB Gasoline A2 (Platts) vs. RBOB Gasoline Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Gulf Coast CBOB Gasoline A2 (Platts) vs. RBOB Gasoline Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Gulf Coast CBOB Gasoline A2 (Platts) vs. RBOB Gasoline Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Gulf Coast CBOB Gasoline A2 (Platts) vs. RBOB Gasoline Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Gulf Coast CBOB Gasoline A2 (Platts) vs. RBOB Gasoline Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Gulf Coast CBOB Gasoline A2 (Platts) vs. RBOB Gasoline Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Gulf Coast CBOB Gasoline A2 (Platts) vs. RBOB Gasoline Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Gulf Coast CBOB Gasoline A2 (Platts) vs. RBOB Gasoline Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

CSW

Introduction

This page shows the trading hours, holidays, and time zone of the CSW contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the CSW contract in the NYMEX Future market:

Weekday	Time (America/New York)
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the CSW contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the CSW contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04

2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19

2023-07-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-20	2025-04-18

Early Closes

The following table shows the early closes for the CSW contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/New York)**
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00

2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00

2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00

2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The CSW contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

CSX

Introduction

This page shows the trading hours, holidays, and time zone of the WTI Financial Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the WTI Financial Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the WTI Financial Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the WTI Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the WTI Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the WTI Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00

2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00

2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00

2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00

2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00
2024-02-19	14:30:00

2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00

2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the WTI Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00

2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The WTI Financial Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

CU

Introduction

This page shows the trading hours, holidays, and time zone of the Chicago Ethanol (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Chicago Ethanol (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Chicago Ethanol (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Chicago Ethanol (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Chicago Ethanol (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Chicago Ethanol (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00

2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00

2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00

2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00

2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00
2024-02-19	14:30:00

2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00

2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Chicago Ethanol (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00

2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Chicago Ethanol (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

D1N

Introduction

This page shows the trading hours, holidays, and time zone of the Singapore Mogas 92 Unleaded (Platts) Brent Crack Spread Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Singapore Mogas 92 Unleaded (Platts) Brent Crack Spread Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Singapore Mogas 92 Unleaded (Platts) Brent Crack Spread Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Singapore Mogas 92 Unleaded (Platts) Brent Crack Spread Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Singapore Mogas 92 Unleaded (Platts) Brent Crack Spread Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Singapore Mogas 92 Unleaded (Platts) Brent Crack Spread Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Singapore Mogas 92 Unleaded (Platts) Brent Crack Spread Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Singapore Mogas 92 Unleaded (Platts) Brent Crack Spread Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

DCB

Introduction

This page shows the trading hours, holidays, and time zone of the Dubai Crude Oil (Platts) Financial Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Dubai Crude Oil (Platts) Financial Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Dubai Crude Oil (Platts) Financial Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Dubai Crude Oil (Platts) Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Dubai Crude Oil (Platts) Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Dubai Crude Oil (Platts) Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Dubai Crude Oil (Platts) Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00

2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00

2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00

2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00

2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Dubai Crude Oil (Platts) Financial Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

E6

Introduction

This page shows the trading hours, holidays, and time zone of the Japan C&F Naphtha (Platts) BALMO Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Japan C&F Naphtha (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Japan C&F Naphtha (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Japan C&F Naphtha (Platts) BALMO Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Japan C&F Naphtha (Platts) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Japan C&F Naphtha (Platts) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Japan C&F Naphtha (Platts) BALMO Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Japan C&F Naphtha (Platts) BALMO Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

EN

Introduction

This page shows the trading hours, holidays, and time zone of the European Naphtha (Platts) Crack Spread Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the European Naphtha (Platts) Crack Spread Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the European Naphtha (Platts) Crack Spread Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the European Naphtha (Platts) Crack Spread Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the European Naphtha (Platts) Crack Spread Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the European Naphtha (Platts) Crack Spread Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the European Naphtha (Platts) Crack Spread Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The European Naphtha (Platts) Crack Spread Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

EPN

Introduction

This page shows the trading hours, holidays, and time zone of the European Propane CIF ARA (Argus) vs. Naphtha Cargoes CIF NWE (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the European Propane CIF ARA (Argus) vs. Naphtha Cargoes CIF NWE (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the European Propane CIF ARA (Argus) vs. Naphtha Cargoes CIF NWE (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the European Propane CIF ARA (Argus) vs. Naphtha Cargoes CIF NWE (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the European Propane CIF ARA (Argus) vs. Naphtha Cargoes CIF NWE (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the European Propane CIF ARA (Argus) vs. Naphtha Cargoes CIF NWE (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the European Propane CIF ARA (Argus) vs. Naphtha Cargoes CIF NWE (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The European Propane CIF ARA (Argus) vs. Naphtha Cargoes CIF NWE (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

EVC

Introduction

This page shows the trading hours, holidays, and time zone of the Singapore Fuel Oil 380 cst (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Singapore Fuel Oil 380 cst (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Singapore Fuel Oil 380 cst (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Singapore Fuel Oil 380 cst (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Singapore Fuel Oil 380 cst (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Singapore Fuel Oil 380 cst (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Singapore Fuel Oil 380 cst (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Singapore Fuel Oil 380 cst (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

EWG

Introduction

This page shows the trading hours, holidays, and time zone of the East-West Gasoline Spread (Platts-Argus) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the East-West Gasoline Spread (Platts-Argus) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the East-West Gasoline Spread (Platts-Argus) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the East-West Gasoline Spread (Platts-Argus) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the East-West Gasoline Spread (Platts-Argus) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the East-West Gasoline Spread (Platts-Argus) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the East-West Gasoline Spread (Platts-Argus) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The East-West Gasoline Spread (Platts-Argus) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

EWN

Introduction

This page shows the trading hours, holidays, and time zone of the East-West Naphtha: Japan C&F vs. Cargoes CIF NWE Spread (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the East-West Naphtha: Japan C&F vs. Cargoes CIF NWE Spread (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the East-West Naphtha: Japan C&F vs. Cargoes CIF NWE Spread (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the East-West Naphtha: Japan C&F vs. Cargoes CIF NWE Spread (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the East-West Naphtha: Japan C&F vs. Cargoes CIF NWE Spread (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the East-West Naphtha: Japan C&F vs. Cargoes CIF NWE Spread (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the East-West Naphtha: Japan C&F vs. Cargoes CIF NWE Spread (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The East-West Naphtha: Japan C&F vs. Cargoes CIF NWE Spread (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

EXR

Introduction

This page shows the trading hours, holidays, and time zone of the RBOB Gasoline vs. Euro-bob Oxy NWE Barges (Argus) (350000 gallons) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the RBOB Gasoline vs. Euro-bob Oxy NWE Barges (Argus) (350000 gallons) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the RBOB Gasoline vs. Euro-bob Oxy NWE Barges (Argus) (350000 gallons) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the RBOB Gasoline vs. Euro-bob Oxy NWE Barges (Argus) (350000 gallons) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the RBOB Gasoline vs. Euro-bob Oxy NWE Barges (Argus) (350000 gallons) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the RBOB Gasoline vs. Euro-bob Oxy NWE Barges (Argus) (350000 gallons) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the RBOB Gasoline vs. Euro-bob Oxy NWE Barges (Argus) (350000 gallons) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The RBOB Gasoline vs. Euro-bob Oxy NWE Barges (Argus) (350000 gallons) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

FO

Introduction

This page shows the trading hours, holidays, and time zone of the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

FRC

Introduction

This page shows the trading hours, holidays, and time zone of the Freight Route TC14 (Baltic) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Freight Route TC14 (Baltic) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Freight Route TC14 (Baltic) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Freight Route TC14 (Baltic) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Freight Route TC14 (Baltic) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Freight Route TC14 (Baltic) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Freight Route TC14 (Baltic) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00

2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00

2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00

2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00

2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Freight Route TC14 (Baltic) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

FSS

Introduction

This page shows the trading hours, holidays, and time zone of the 1% Fuel Oil Cargoes FOB NWE (Platts) vs. 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the 1% Fuel Oil Cargoes FOB NWE (Platts) vs. 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the 1% Fuel Oil Cargoes FOB NWE (Platts) vs. 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the 1% Fuel Oil Cargoes FOB NWE (Platts) vs. 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the 1% Fuel Oil Cargoes FOB NWE (Platts) vs. 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the 1% Fuel Oil Cargoes FOB NWE (Platts) vs. 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the 1% Fuel Oil Cargoes FOB NWE (Platts) vs. 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The 1% Fuel Oil Cargoes FOB NWE (Platts) vs. 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

GCU

Introduction

This page shows the trading hours, holidays, and time zone of the Gulf Coast HSFO (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Gulf Coast HSFO (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Gulf Coast HSFO (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Gulf Coast HSFO (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Gulf Coast HSFO (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Gulf Coast HSFO (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Gulf Coast HSFO (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Gulf Coast HSFO (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

HCL

Introduction

This page shows the trading hours, holidays, and time zone of the WTI Houston Crude Oil Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the WTI Houston Crude Oil Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the WTI Houston Crude Oil Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the WTI Houston Crude Oil Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04

2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19

2023-07-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-20	2025-04-18

Early Closes

The following table shows the early closes for the WTI Houston Crude Oil Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/New York)**
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00

2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00

2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00

2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The WTI Houston Crude Oil Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

HH

Introduction

This page shows the trading hours, holidays, and time zone of the Natural Gas (Henry Hub) Last-day Financial Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Natural Gas (Henry Hub) Last-day Financial Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Natural Gas (Henry Hub) Last-day Financial Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Natural Gas (Henry Hub) Last-day Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Natural Gas (Henry Hub) Last-day Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Natural Gas (Henry Hub) Last-day Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Natural Gas (Henry Hub) Last-day Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Natural Gas (Henry Hub) Last-day Financial Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

HO

Introduction

This page shows the trading hours, holidays, and time zone of the NY Harbor ULSD Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the NY Harbor ULSD Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NY Harbor ULSD Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the NY Harbor ULSD Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00

2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00

2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00

2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00

2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00
2024-02-19	14:30:00

2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00

2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the NY Harbor ULSD Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00

2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The NY Harbor ULSD Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

HP

Introduction

This page shows the trading hours, holidays, and time zone of the Natural Gas (Henry Hub) Penultimate Financial Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Natural Gas (Henry Hub) Penultimate Financial Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Natural Gas (Henry Hub) Penultimate Financial Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Natural Gas (Henry Hub) Penultimate Financial Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Natural Gas (Henry Hub) Penultimate Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Natural Gas (Henry Hub) Penultimate Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Natural Gas (Henry Hub) Penultimate Financial Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Natural Gas (Henry Hub) Penultimate Financial Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

HRC

Introduction

This page shows the trading hours, holidays, and time zone of the U.S. Midwest Domestic Hot-Rolled Coil Steel (CRU) Index Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the U.S. Midwest Domestic Hot-Rolled Coil Steel (CRU) Index Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the U.S. Midwest Domestic Hot-Rolled Coil Steel (CRU) Index Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the U.S. Midwest Domestic Hot-Rolled Coil Steel (CRU) Index Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04

2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19

2023-07-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-20	2025-04-18

Early Closes

The following table shows the early closes for the U.S. Midwest Domestic Hot-Rolled Coil Steel (CRU) Index Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/New York)**
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00

2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00

2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00

2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00

Late Opens

There are no days with late opens.

Time Zone

The U.S. Midwest Domestic Hot-Rolled Coil Steel (CRU) Index Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

HTT

Introduction

This page shows the trading hours, holidays, and time zone of the WTI Houston (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the WTI Houston (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the WTI Houston (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the WTI Houston (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the WTI Houston (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the WTI Houston (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the WTI Houston (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The WTI Houston (Argus) vs. WTI Trade Month Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

M1B

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Gasoil 0.1% Barges FOB ARA (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Gasoil 0.1% Barges FOB ARA (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Gasoil 0.1% Barges FOB ARA (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Gasoil 0.1% Barges FOB ARA (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro Gasoil 0.1% Barges FOB ARA (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro Gasoil 0.1% Barges FOB ARA (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Micro Gasoil 0.1% Barges FOB ARA (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Micro Gasoil 0.1% Barges FOB ARA (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

M35

Introduction

This page shows the trading hours, holidays, and time zone of the Micro European 3.5% Fuel Oil Cargoes FOB Med (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro European 3.5% Fuel Oil Cargoes FOB Med (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro European 3.5% Fuel Oil Cargoes FOB Med (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro European 3.5% Fuel Oil Cargoes FOB Med (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro European 3.5% Fuel Oil Cargoes FOB Med (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro European 3.5% Fuel Oil Cargoes FOB Med (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Micro European 3.5% Fuel Oil Cargoes FOB Med (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Micro European 3.5% Fuel Oil Cargoes FOB Med (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

M5F

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Coal (API 5) fob Newcastle (Argus/McCloskey) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Coal (API 5) fob Newcastle (Argus/McCloskey) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Coal (API 5) fob Newcastle (Argus/McCloskey) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Coal (API 5) fob Newcastle (Argus/McCloskey) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the NYMEX Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04

2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19

2023-07-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-20	2025-04-18

Early Closes

The following table shows the early closes for the Micro Coal (API 5) fob Newcastle (Argus/McCloskey) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/New York)**
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00

2010-11-25	13:15:00
2010-11-26	13:45:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2011-12-31	16:15:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00

2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00
2016-12-23	17:00:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00

2017-11-23	13:00:00
2017-11-24	13:45:00
2017-12-22	17:00:00
2017-12-26	17:00:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	12:00:00
2020-02-17	12:00:00
2020-05-25	12:00:00
2020-07-03	12:00:00
2020-09-07	12:00:00
2020-11-26	12:00:00
2020-11-27	12:45:00
2020-12-24	12:45:00

2021-01-18	12:00:00
2021-02-15	12:00:00
2021-05-31	12:00:00
2021-07-05	12:00:00
2021-09-06	12:00:00
2021-11-25	12:00:00
2021-11-26	12:45:00
2022-01-17	13:30:00
2022-02-21	13:30:00
2022-05-30	13:30:00
2022-06-20	13:30:00
2022-07-04	13:30:00
2022-09-05	12:00:00
2022-11-24	13:30:00
2022-11-25	12:45:00

Late Opens

There are no days with late opens.

Time Zone

The Micro Coal (API 5) fob Newcastle (Argus/McCloskey) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

MAF

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Singapore Fuel Oil 380CST (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Singapore Fuel Oil 380CST (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Singapore Fuel Oil 380CST (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Singapore Fuel Oil 380CST (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro Singapore Fuel Oil 380CST (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro Singapore Fuel Oil 380CST (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Micro Singapore Fuel Oil 380CST (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Micro Singapore Fuel Oil 380CST (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

MCL

Introduction

This page shows the trading hours, holidays, and time zone of the Micro WTI Crude Oil Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro WTI Crude Oil Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro WTI Crude Oil Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro WTI Crude Oil Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro WTI Crude Oil Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro WTI Crude Oil Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00

2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00

2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00

2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00

2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00
2024-02-19	14:30:00

2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00

2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Micro WTI Crude Oil Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00

2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Micro WTI Crude Oil Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

MEF

Introduction

This page shows the trading hours, holidays, and time zone of the Micro European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Micro European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Micro European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

NG

Introduction

This page shows the trading hours, holidays, and time zone of the Henry Hub Natural Gas Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Henry Hub Natural Gas Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Henry Hub Natural Gas Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Henry Hub Natural Gas Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Henry Hub Natural Gas Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Henry Hub Natural Gas Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00

2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00

2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00

2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00

2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00
2024-02-19	14:30:00

2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00

2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Henry Hub Natural Gas Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00

2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Henry Hub Natural Gas Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

PA

Introduction

This page shows the trading hours, holidays, and time zone of the Palladium Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Palladium Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Palladium Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Palladium Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Palladium Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Palladium Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2024-01-15	14:30:00
2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00

2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Palladium Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Palladium Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

PAM

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Palladium Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Palladium Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Palladium Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Palladium Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro Palladium Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro Palladium Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00

2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00

2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00

2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00

2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00
2024-02-19	14:30:00

2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00
2026-11-27	14:45:00

2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Micro Palladium Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00

2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00

2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00

2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00

2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Micro Palladium Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

PL

Introduction

This page shows the trading hours, holidays, and time zone of the Platinum Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Platinum Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Platinum Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Platinum Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Platinum Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Platinum Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2024-01-15	14:30:00
2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00

2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Platinum Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Open (America/New York)
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Platinum Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

R5O

Introduction

This page shows the trading hours, holidays, and time zone of the Micro European FOB Rdam Marine Fuel 0.5% Barges (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro European FOB Rdam Marine Fuel 0.5% Barges (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro European FOB Rdam Marine Fuel 0.5% Barges (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro European FOB Rdam Marine Fuel 0.5% Barges (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro European FOB Rdam Marine Fuel 0.5% Barges (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro European FOB Rdam Marine Fuel 0.5% Barges (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Micro European FOB Rdam Marine Fuel 0.5% Barges (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Micro European FOB Rdam Marine Fuel 0.5% Barges (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

RB

Introduction

This page shows the trading hours, holidays, and time zone of the RBOB Gasoline Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the RBOB Gasoline Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the RBOB Gasoline Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the RBOB Gasoline Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the RBOB Gasoline Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the RBOB Gasoline Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00
2010-05-28	16:15:00

2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00
2013-05-27	13:15:00

2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00
2016-11-25	13:45:00

2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00
2020-09-07	13:00:00

2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00
2024-02-19	14:30:00

2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00
2026-11-26	14:30:00

2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the RBOB Gasoline Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00
2012-07-04	18:00:00
2012-09-03	18:00:00

2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00
2017-07-04	18:00:00
2017-09-04	18:00:00

2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00
2022-02-21	18:00:00
2022-05-30	18:00:00

2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00
2025-09-01	18:00:00
2025-11-27	18:00:00

2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The RBOB Gasoline Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

S5O

Introduction

This page shows the trading hours, holidays, and time zone of the Micro Singapore FOB Marine Fuel 0.5% (Platts) Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the Micro Singapore FOB Marine Fuel 0.5% (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 09:30:00
Tuesday	00:00:00 to 09:30:00
Wednesday	00:00:00 to 09:30:00
Thursday	00:00:00 to 09:30:00
Friday	00:00:00 to 09:30:00

Regular Trading Hours

The following table shows the regular trading hours for the Micro Singapore FOB Marine Fuel 0.5% (Platts) Futures contract in the NYMEX Future market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 17:00:00
Tuesday	09:30:00 to 17:00:00
Wednesday	09:30:00 to 17:00:00
Thursday	09:30:00 to 17:00:00
Friday	09:30:00 to 17:00:00

Post-market Hours

The following table shows the post-market hours for the Micro Singapore FOB Marine Fuel 0.5% (Platts) Futures contract in the NYMEX Future market:

Weekday	Time (America/New York)
Monday	18:00:00 to 24:00:00
Tuesday	18:00:00 to 24:00:00
Wednesday	18:00:00 to 24:00:00
Thursday	18:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the Micro Singapore FOB Marine Fuel 0.5% (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the Micro Singapore FOB Marine Fuel 0.5% (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2009-01-19	17:15:00
2009-02-16	17:15:00
2009-05-22	16:15:00
2009-05-25	13:15:00
2009-07-03	13:30:00
2009-09-04	16:15:00
2009-09-07	13:15:00
2009-10-09	16:15:00
2009-11-26	13:15:00
2009-11-27	13:45:00
2009-12-24	13:45:00
2010-01-15	16:15:00
2010-01-18	13:15:00
2010-02-12	16:15:00
2010-02-15	13:15:00

2010-05-28	16:15:00
2010-05-31	13:15:00
2010-07-02	16:15:00
2010-07-05	13:15:00
2010-09-03	16:15:00
2010-09-06	13:15:00
2010-10-08	16:15:00
2010-11-25	13:15:00
2010-11-26	13:45:00
2010-12-31	16:15:00
2011-01-14	16:15:00
2011-01-17	13:15:00
2011-02-21	13:15:00
2011-05-30	13:15:00
2011-07-04	13:15:00
2011-09-05	13:15:00
2011-11-24	13:15:00
2011-11-25	13:45:00
2012-01-16	13:15:00
2012-02-20	13:15:00
2012-05-28	13:15:00
2012-07-04	13:15:00
2012-09-03	13:15:00
2012-11-22	13:15:00
2012-11-23	13:45:00
2012-12-24	13:45:00
2013-01-21	13:15:00
2013-02-18	13:15:00

2013-05-27	13:15:00
2013-07-04	13:15:00
2013-09-02	13:15:00
2013-11-28	13:15:00
2013-11-29	13:45:00
2013-12-24	13:45:00
2014-01-20	13:15:00
2014-02-17	13:15:00
2014-05-26	13:00:00
2014-07-04	13:00:00
2014-09-01	13:00:00
2014-11-27	13:00:00
2014-11-28	13:45:00
2014-12-24	13:45:00
2015-01-19	13:00:00
2015-02-16	13:00:00
2015-05-25	13:00:00
2015-07-03	13:00:00
2015-09-07	13:00:00
2015-11-26	13:00:00
2015-11-27	13:45:00
2015-12-24	13:45:00
2016-01-18	13:00:00
2016-02-15	13:00:00
2016-05-30	13:00:00
2016-07-04	13:00:00
2016-09-05	13:00:00
2016-11-24	13:00:00

2016-11-25	13:45:00
2017-01-16	13:00:00
2017-02-20	13:00:00
2017-05-29	13:00:00
2017-07-04	13:00:00
2017-09-04	13:00:00
2017-11-23	13:00:00
2017-11-24	13:45:00
2018-01-15	13:00:00
2018-02-19	13:00:00
2018-05-28	13:00:00
2018-07-04	13:00:00
2018-09-03	13:00:00
2018-11-22	13:00:00
2018-11-23	13:45:00
2018-12-24	13:45:00
2019-01-21	13:00:00
2019-02-18	13:00:00
2019-05-27	13:00:00
2019-07-04	13:00:00
2019-09-02	13:00:00
2019-11-28	13:00:00
2019-11-29	13:45:00
2019-12-24	13:45:00
2020-01-20	13:00:00
2020-02-17	13:00:00
2020-05-25	13:00:00
2020-07-03	13:00:00

2020-09-07	13:00:00
2020-11-26	13:00:00
2020-11-27	13:45:00
2020-12-24	13:45:00
2021-01-18	13:00:00
2021-02-15	13:00:00
2021-05-31	13:00:00
2021-07-05	13:00:00
2021-09-06	13:00:00
2021-11-25	13:00:00
2021-11-26	13:45:00
2022-01-17	14:30:00
2022-02-21	14:30:00
2022-05-30	14:30:00
2022-06-20	14:30:00
2022-07-04	14:30:00
2022-09-05	14:30:00
2022-11-24	14:30:00
2022-11-25	13:45:00
2023-01-16	14:30:00
2023-02-20	14:30:00
2023-05-29	14:30:00
2023-06-19	14:30:00
2023-07-04	14:30:00
2023-09-04	14:30:00
2023-11-23	14:30:00
2023-11-24	13:45:00
2024-01-15	14:30:00

2024-02-19	14:30:00
2024-03-28	17:00:00
2024-05-27	14:30:00
2024-06-19	14:30:00
2024-07-04	14:30:00
2024-09-02	14:30:00
2024-11-28	14:30:00
2024-12-24	13:45:00
2024-12-31	17:00:00
2025-01-20	14:30:00
2025-02-17	14:30:00
2025-04-17	17:00:00
2025-05-26	14:30:00
2025-06-19	14:30:00
2025-07-04	13:00:00
2025-09-01	14:30:00
2025-11-27	14:30:00
2025-11-28	14:45:00
2025-12-24	13:45:00
2025-12-25	17:00:00
2025-12-31	17:00:00
2026-01-19	14:30:00
2026-02-16	14:30:00
2026-04-02	17:00:00
2026-05-25	14:30:00
2026-06-19	13:00:00
2026-07-03	13:00:00
2026-09-07	14:30:00

2026-11-26	14:30:00
2026-11-27	14:45:00
2026-12-24	13:45:00
2026-12-31	17:00:00

Late Opens

The following table shows the late opens for the Micro Singapore FOB Marine Fuel 0.5% (Platts) Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/New York)**
2009-01-19	18:00:00
2009-02-16	18:00:00
2009-05-25	18:00:00
2009-09-07	18:00:00
2009-11-26	18:00:00
2010-01-18	18:00:00
2010-02-15	18:00:00
2010-05-31	18:00:00
2010-07-05	18:00:00
2010-09-06	18:00:00
2010-11-25	18:00:00
2011-01-17	18:00:00
2011-02-21	18:00:00
2011-05-30	18:00:00
2011-07-04	18:00:00
2011-09-05	18:00:00
2011-11-24	18:00:00
2012-01-16	18:00:00
2012-02-20	18:00:00
2012-05-28	18:00:00

2012-07-04	18:00:00
2012-09-03	18:00:00
2012-11-22	18:00:00
2013-01-21	18:00:00
2013-02-18	18:00:00
2013-05-27	18:00:00
2013-07-04	18:00:00
2013-09-02	18:00:00
2013-11-28	18:00:00
2014-01-20	18:00:00
2014-02-17	18:00:00
2014-05-26	18:00:00
2014-09-01	18:00:00
2014-11-27	18:00:00
2015-01-19	18:00:00
2015-02-16	18:00:00
2015-05-25	18:00:00
2015-09-07	18:00:00
2015-11-26	18:00:00
2016-01-18	18:00:00
2016-02-15	18:00:00
2016-05-30	18:00:00
2016-07-04	18:00:00
2016-09-05	18:00:00
2016-11-24	18:00:00
2017-01-16	18:00:00
2017-02-20	18:00:00
2017-05-29	18:00:00

2017-07-04	18:00:00
2017-09-04	18:00:00
2017-11-23	18:00:00
2018-01-15	18:00:00
2018-02-19	18:00:00
2018-05-28	18:00:00
2018-07-04	18:00:00
2018-09-03	18:00:00
2018-11-22	18:00:00
2019-01-21	18:00:00
2019-02-18	18:00:00
2019-05-27	18:00:00
2019-07-04	18:00:00
2019-09-02	18:00:00
2019-11-28	18:00:00
2020-01-20	18:00:00
2020-02-17	18:00:00
2020-05-25	18:00:00
2020-07-03	18:00:00
2020-09-07	18:00:00
2020-11-26	18:00:00
2021-01-18	18:00:00
2021-02-15	18:00:00
2021-05-31	18:00:00
2021-07-05	18:00:00
2021-09-06	18:00:00
2021-11-25	18:00:00
2022-01-17	18:00:00

2022-02-21	18:00:00
2022-05-30	18:00:00
2022-06-20	18:00:00
2022-07-04	18:00:00
2022-09-05	18:00:00
2022-11-24	18:00:00
2023-01-16	18:00:00
2023-02-20	18:00:00
2023-05-29	18:00:00
2023-06-19	18:00:00
2023-07-04	18:00:00
2023-09-04	18:00:00
2023-11-23	18:00:00
2023-12-25	18:00:00
2024-01-01	18:00:00
2024-01-15	18:00:00
2024-02-19	18:00:00
2024-05-27	18:00:00
2024-06-19	18:00:00
2024-07-04	18:00:00
2024-09-02	18:00:00
2024-11-28	18:00:00
2024-12-25	18:00:00
2025-01-01	18:00:00
2025-01-20	18:00:00
2025-02-17	18:00:00
2025-05-26	18:00:00
2025-06-19	18:00:00

2025-09-01	18:00:00
2025-11-27	18:00:00
2025-12-25	18:00:00
2026-01-01	18:00:00
2026-01-19	18:00:00
2026-02-16	18:00:00
2026-05-25	18:00:00
2026-09-07	18:00:00
2026-11-26	18:00:00

Time Zone

The Micro Singapore FOB Marine Fuel 0.5% (Platts) Futures contract in the NYMEX Future market trades in the `America/New York` time zone.

NYMEX

YO

Introduction

This page shows the trading hours, holidays, and time zone of the No. 11 Sugar Futures contract in the NYMEX Future market.

Pre-market Hours

The following table shows the pre-market hours for the No. 11 Sugar Futures contract in the NYMEX Future market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 08:30:00
Tuesday	00:00:00 to 08:30:00
Wednesday	00:00:00 to 08:30:00
Thursday	00:00:00 to 08:30:00
Friday	00:00:00 to 08:30:00

Regular Trading Hours

The following table shows the regular trading hours for the No. 11 Sugar Futures contract in the NYMEX Future market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 16:00:00
Tuesday	08:30:00 to 16:00:00
Wednesday	08:30:00 to 16:00:00
Thursday	08:30:00 to 16:00:00
Friday	08:30:00 to 16:00:00

Post-market Hours

The following table shows the post-market hours for the No. 11 Sugar Futures contract in the NYMEX Future market:

Weekday	Time (America/Chicago)
Monday	17:00:00 to 24:00:00
Tuesday	17:00:00 to 24:00:00
Wednesday	17:00:00 to 24:00:00
Thursday	17:00:00 to 24:00:00

Holidays

The following table shows the dates of holidays for the No. 11 Sugar Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)		
2026-04-03	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the No. 11 Sugar Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/Chicago)
2010-12-31	12:15:00
2013-03-28	13:55:00
2013-07-03	12:00:00
2013-11-27	15:05:00
2013-11-29	12:00:00
2013-12-24	12:00:00
2014-01-19	16:00:00
2014-02-16	16:00:00
2014-04-17	13:55:00
2014-04-20	16:00:00
2014-05-25	16:00:00
2014-07-03	12:00:00
2014-11-28	12:00:00
2014-12-24	12:00:00
2015-04-02	13:55:00
2015-07-02	12:00:00

2015-11-27	12:00:00
2015-12-24	12:00:00
2015-12-31	13:55:00
2016-03-24	13:55:00
2016-11-25	12:00:00
2016-12-23	12:00:00
2017-04-13	13:55:00
2017-07-03	12:00:00
2017-11-24	12:00:00
2017-12-22	12:00:00
2018-03-29	13:55:00
2018-07-03	12:00:00
2018-11-23	12:00:00
2018-12-24	12:00:00
2018-12-31	13:55:00
2019-04-18	13:55:00
2019-07-03	12:05:00
2019-11-29	12:05:00
2019-12-24	12:05:00
2019-12-31	13:55:00
2020-07-02	12:05:00
2020-11-27	12:05:00
2020-12-24	12:05:00
2021-11-26	12:05:00
2022-11-25	12:05:00
2023-11-24	12:05:00
2024-03-28	15:05:00
2024-11-27	15:05:00

2024-12-31	16:00:00
2025-01-19	16:00:00
2025-02-16	16:00:00
2025-04-17	16:00:00
2025-05-25	16:00:00
2025-06-18	16:00:00
2025-07-04	12:00:00
2025-08-31	16:00:00
2025-11-26	16:00:00
2025-11-28	12:05:00
2025-12-24	12:05:00
2025-12-31	16:00:00
2026-01-18	16:00:00
2026-02-15	16:00:00
2026-04-02	16:00:00
2026-05-24	16:00:00
2026-06-19	12:00:00
2026-07-03	12:00:00
2026-09-06	16:00:00
2026-11-25	16:00:00
2026-11-27	13:45:00
2026-12-24	12:45:00
2026-12-31	16:00:00

Late Opens

The following table shows the late opens for the No. 11 Sugar Futures contract in the NYMEX Future market:

Date (*yyyy-mm-dd*)	**Time Of Market Open (America/Chicago)**
2011-11-24	17:00:00
2011-12-27	09:00:00

2012-01-03	09:00:00
2012-01-17	09:00:00
2012-02-21	09:00:00
2012-05-29	09:00:00
2012-07-05	09:00:00
2012-09-04	09:00:00
2012-11-22	17:00:00
2012-12-26	09:00:00
2013-01-02	09:00:00
2013-01-22	09:00:00
2013-02-19	09:00:00
2013-05-28	09:00:00
2013-07-05	09:00:00
2013-09-03	09:00:00
2013-11-29	09:00:00
2013-12-26	09:00:00
2014-01-02	09:00:00
2014-01-21	09:00:00
2014-02-18	09:00:00
2014-04-21	09:00:00
2014-05-27	09:00:00
2014-07-07	09:00:00
2014-09-02	09:00:00
2014-11-28	09:00:00
2014-12-26	09:00:00
2015-01-02	09:00:00
2015-01-20	09:00:00
2015-02-17	09:00:00

2015-04-06	09:00:00
2015-05-26	09:00:00
2015-07-06	09:00:00
2015-09-08	09:00:00
2015-11-27	09:00:00
2015-12-28	09:00:00
2016-01-04	09:00:00
2016-01-19	09:00:00
2016-02-16	09:00:00
2016-03-28	09:00:00
2016-05-31	09:00:00
2016-07-05	09:00:00
2016-09-06	09:00:00
2016-11-25	09:00:00
2017-01-16	09:00:00
2017-02-20	09:00:00
2017-04-17	09:00:00
2017-05-30	09:00:00
2017-07-05	09:00:00
2017-09-05	09:00:00
2018-01-16	09:00:00
2018-02-20	09:00:00
2018-04-02	09:00:00
2018-05-29	09:00:00
2018-07-05	09:00:00
2018-09-04	09:00:00
2019-01-22	09:00:00
2019-02-19	09:00:00

2019-04-22	09:00:00
2019-05-28	09:00:00
2019-07-05	09:00:00
2019-09-03	09:00:00
2020-01-21	09:00:00
2020-02-18	09:00:00
2020-04-13	09:00:00
2020-05-26	09:00:00
2020-07-06	09:00:00
2020-09-08	09:00:00
2021-01-19	09:00:00
2021-02-16	09:00:00
2021-04-05	09:00:00
2021-06-01	09:00:00
2025-01-01	17:00:00
2025-01-20	17:00:00
2025-02-17	17:00:00
2025-05-26	17:00:00
2025-06-19	17:00:00
2025-09-01	17:00:00
2025-11-27	17:00:00
2025-12-25	17:00:00
2026-01-01	17:00:00
2026-01-19	17:00:00
2026-02-16	17:00:00
2026-05-25	17:00:00
2026-09-07	17:00:00
2026-11-26	17:00:00

Time Zone

The No. 11 Sugar Futures contract in the NYMEX Future market trades in the `America/Chicago` time zone.

Market Hours

NYSELIFFE

Introduction

This page shows the trading hours, holidays, and time zone of the NYSELIFFE Future market.

Historical data for backtesting is unavailable for NYSELIFFE. In live trading, LEAN sources this data from your brokerage or a third-party data provider .

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NYSELIFFE Future market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 24:00:00
Tuesday	00:00:00 to 24:00:00
Wednesday	00:00:00 to 24:00:00
Thursday	00:00:00 to 24:00:00
Friday	00:00:00 to 18:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the NYSELIFFE Future market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23

2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03

2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-25	2016-12-26	2017-01-02	2017-01-16
2017-02-20	2017-04-14	2017-05-29	2017-07-04	2017-09-04
2017-11-23	2017-12-25	2018-01-01	2018-01-15	2018-02-19
2018-03-30	2018-05-28	2018-07-04	2018-09-03	2018-11-22
2018-12-05	2018-12-25	2019-01-01	2019-01-21	2019-02-18
2019-04-19	2019-05-27	2019-07-04	2019-09-02	2019-11-28
2019-12-25	2020-01-01	2020-01-20	2020-02-17	2020-04-10
2020-05-25	2020-07-03	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-07-05	2021-09-06	2021-11-25	2021-12-24	2022-01-01
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-06-20
2022-07-04	2022-09-05	2022-11-24	2022-12-26	2023-01-02
2023-01-16	2023-02-20	2023-04-07	2023-05-29	2023-06-19
2023-07-04	2023-09-04	2023-11-23	2023-12-25	2024-01-01
2024-01-15	2024-02-19	2024-03-29	2024-05-27	2024-06-19
2024-07-04	2024-09-02	2024-11-28	2024-12-25	2025-01-01
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-07-04
2025-09-01	2025-11-27	2025-12-25	2026-01-01	2026-01-19
2026-02-16	2026-04-03	2026-05-25	2026-06-19	2026-07-03
2026-09-07	2026-11-26	2026-12-25	2027-01-01	

Early Closes

The following table shows the early closes for the NYSELIFFE Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (America/New York)
2023-07-03	18:00:00
2023-11-24	18:00:00
2024-07-03	18:00:00
2024-11-29	18:00:00
2024-12-24	18:00:00
2025-07-03	18:00:00
2025-11-28	18:00:00
2025-12-24	18:00:00
2026-11-27	18:00:00
2026-12-24	18:00:00

Late Opens

There are no days with late opens.

Time Zone

The NYSELIFFE Future market trades in the `America/New York` time zone.

Market Hours

SGX

Introduction

This page shows the trading hours, holidays, and time zone of the SGX Future market.

Historical data for backtesting is unavailable for SGX. In live trading, LEAN sources this data from your brokerage or a third-party data provider .

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the SGX Future market:

Weekday	**Time (Asia/Singapore)**
Monday	09:00:00 to 12:00:00, 13:00:00 to 17:00:00
Tuesday	09:00:00 to 12:00:00, 13:00:00 to 17:00:00
Wednesday	09:00:00 to 12:00:00, 13:00:00 to 17:00:00
Thursday	09:00:00 to 12:00:00, 13:00:00 to 17:00:00
Friday	09:00:00 to 12:00:00, 13:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the SGX Future market:

Date (*yyyy-mm-dd*)				
2016-01-01	2016-02-08	2016-02-09	2016-03-25	2016-05-01
2016-05-21	2016-07-06	2016-08-09	2016-09-12	2016-10-29
2016-12-25	2017-01-01	2017-01-28	2017-01-29	2017-04-14
2017-05-01	2017-05-10	2017-06-25	2017-08-09	2017-09-01
2017-10-18	2017-12-25	2018-01-01	2018-02-16	2018-02-17
2018-03-30	2018-05-01	2018-05-29	2018-06-15	2018-08-09
2018-08-22	2018-11-06	2018-12-25	2019-01-01	2019-02-05
2019-02-06	2019-04-19	2019-05-01	2019-05-20	2019-06-05
2019-08-09	2019-08-12	2019-10-28	2019-12-25	2020-01-01
2020-01-27	2020-04-10	2020-05-01	2020-05-07	2020-05-25
2020-07-31	2020-08-10	2020-11-14	2020-12-25	2021-01-01
2021-02-12	2021-04-02	2021-05-01	2021-08-09	2021-12-25
2022-01-03	2022-01-31	2022-02-01	2022-02-02	2022-04-15
2022-05-02	2022-08-09	2022-12-26	2022-12-31	2025-01-01
2025-04-18	2025-12-25	2026-01-01		

Early Closes

The following table shows the early closes for the SGX Future market:

Date (*yyyy-mm-dd*)	Time Of Market Close (Asia/Singapore)
2018-02-15	12:00:00
2018-12-24	12:00:00
2018-12-31	12:00:00
2019-04-02	12:00:00
2019-12-24	12:00:00
2019-12-31	12:00:00
2020-01-24	12:00:00
2020-12-24	12:00:00
2020-12-31	12:00:00
2021-02-11	12:00:00
2021-12-24	12:00:00
2021-12-31	12:00:00
2022-01-31	12:00:00

Late Opens

There are no days with late opens.

Time Zone

The SGX Future market trades in the `Asia/Singapore` time zone.

Assets With Other Hours

The following table shows the contracts that have different trading periods than the overall SGX Future market:

Symbol	Name
IN	Nifty Indices Futures
NK	SGX Nikkei 225 Index Futures
TW	MSCI Taiwan Index Futures

SGX

IN

Introduction

This page shows the trading hours, holidays, and time zone of the Nifty Indices Futures contract in the SGX Future market.

Historical data for backtesting is unavailable for SGX. In live trading, LEAN sources this data from your brokerage or a third-party data provider .

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Nifty Indices Futures contract in the SGX Future market:

Weekday	**Time (Asia/Singapore)**
Monday	08:45:00 to 18:15:00, 18:30:00 to 24:00:00
Tuesday	00:00:00 to 06:15:00, 08:45:00 to 18:15:00, 18:30:00 to 24:00:00
Wednesday	00:00:00 to 06:15:00, 08:45:00 to 18:15:00, 18:30:00 to 24:00:00
Thursday	00:00:00 to 06:15:00, 08:45:00 to 18:15:00, 18:30:00 to 24:00:00
Friday	00:00:00 to 06:15:00, 08:45:00 to 18:15:00, 18:30:00 to 24:00:00
Saturday	00:00:00 to 06:15:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the SGX Future market:

Date (*yyyy-mm-dd*)				
2016-01-01	2016-02-08	2016-02-09	2016-03-25	2016-05-01
2016-05-21	2016-07-06	2016-08-09	2016-09-12	2016-10-29
2016-12-25	2017-01-01	2017-01-28	2017-01-29	2017-04-14
2017-05-01	2017-05-10	2017-06-25	2017-08-09	2017-09-01
2017-10-18	2017-12-25	2018-01-01	2018-02-16	2018-02-17
2018-03-30	2018-05-01	2018-05-29	2018-06-15	2018-08-09
2018-08-22	2018-11-06	2018-12-25	2019-01-01	2019-02-05
2019-02-06	2019-04-19	2019-05-01	2019-05-20	2019-06-05
2019-08-09	2019-08-12	2019-10-28	2019-12-25	2020-01-01
2020-01-27	2020-04-10	2020-05-01	2020-05-07	2020-05-25
2020-07-31	2020-08-10	2020-11-14	2020-12-25	2021-01-01
2021-02-12	2021-04-02	2021-05-01	2021-08-09	2021-12-25
2022-01-03	2022-01-31	2022-02-01	2022-02-02	2022-04-15
2022-05-02	2022-08-09	2022-12-26	2022-12-31	2025-01-01
2025-04-18	2025-12-25	2026-01-01		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Nifty Indices Futures contract in the SGX Future market trades in the `Asia/Singapore` time zone.

SGX

NK

Introduction

This page shows the trading hours, holidays, and time zone of the SGX Nikkei 225 Index Futures contract in the SGX Future market.

Historical data for backtesting is unavailable for SGX. In live trading, LEAN sources this data from your brokerage or a third-party data provider .

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the SGX Nikkei 225 Index Futures contract in the SGX Future market:

Weekday	**Time (Asia/Singapore)**
Monday	07:15:00 to 14:30:00, 14:45:00 to 24:00:00
Tuesday	00:00:00 to 05:15:00, 07:15:00 to 14:30:00, 14:45:00 to 24:00:00
Wednesday	00:00:00 to 05:15:00, 07:15:00 to 14:30:00, 14:45:00 to 24:00:00
Thursday	00:00:00 to 05:15:00, 07:15:00 to 14:30:00, 14:45:00 to 24:00:00
Friday	00:00:00 to 05:15:00, 07:15:00 to 14:30:00, 14:45:00 to 24:00:00
Saturday	00:00:00 to 05:15:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the SGX Future market:

Date (*yyyy-mm-dd*)				
2016-01-01	2016-02-08	2016-02-09	2016-03-25	2016-05-01
2016-05-21	2016-07-06	2016-08-09	2016-09-12	2016-10-29
2016-12-25	2017-01-01	2017-01-28	2017-01-29	2017-04-14
2017-05-01	2017-05-10	2017-06-25	2017-08-09	2017-09-01
2017-10-18	2017-12-25	2018-01-01	2018-02-16	2018-02-17
2018-03-30	2018-05-01	2018-05-29	2018-06-15	2018-08-09
2018-08-22	2018-11-06	2018-12-25	2019-01-01	2019-02-05
2019-02-06	2019-04-19	2019-05-01	2019-05-20	2019-06-05
2019-08-09	2019-08-12	2019-10-28	2019-12-25	2020-01-01
2020-01-27	2020-04-10	2020-05-01	2020-05-07	2020-05-25
2020-07-31	2020-08-10	2020-11-14	2020-12-25	2021-01-01
2021-02-12	2021-04-02	2021-05-01	2021-08-09	2021-12-25
2022-01-03	2022-01-31	2022-02-01	2022-02-02	2022-04-15
2022-05-02	2022-08-09	2022-12-26	2022-12-31	2025-01-01
2025-04-18	2025-12-25	2026-01-01		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The SGX Nikkei 225 Index Futures contract in the SGX Future market trades in the `Asia/Singapore` time zone.

SGX

TW

Introduction

This page shows the trading hours, holidays, and time zone of the MSCI Taiwan Index Futures contract in the SGX Future market.

Historical data for backtesting is unavailable for SGX. In live trading, LEAN sources this data from your brokerage or a third-party data provider .

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the MSCI Taiwan Index Futures contract in the SGX Future market:

Weekday	**Time (Asia/Singapore)**
Monday	08:30:00 to 13:50:00, 14:05:00 to 24:00:00
Tuesday	00:00:00 to 05:15:00, 08:30:00 to 13:50:00, 14:05:00 to 24:00:00
Wednesday	00:00:00 to 05:15:00, 08:30:00 to 13:50:00, 14:05:00 to 24:00:00
Thursday	00:00:00 to 05:15:00, 08:30:00 to 13:50:00, 14:05:00 to 24:00:00
Friday	00:00:00 to 05:15:00, 08:30:00 to 13:50:00, 14:05:00 to 24:00:00
Saturday	00:00:00 to 05:15:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the SGX Future market:

Date (*yyyy-mm-dd*)				
2016-01-01	2016-02-08	2016-02-09	2016-03-25	2016-05-01
2016-05-21	2016-07-06	2016-08-09	2016-09-12	2016-10-29
2016-12-25	2017-01-01	2017-01-28	2017-01-29	2017-04-14
2017-05-01	2017-05-10	2017-06-25	2017-08-09	2017-09-01
2017-10-18	2017-12-25	2018-01-01	2018-02-16	2018-02-17
2018-03-30	2018-05-01	2018-05-29	2018-06-15	2018-08-09
2018-08-22	2018-11-06	2018-12-25	2019-01-01	2019-02-05
2019-02-06	2019-04-19	2019-05-01	2019-05-20	2019-06-05
2019-08-09	2019-08-12	2019-10-28	2019-12-25	2020-01-01
2020-01-27	2020-04-10	2020-05-01	2020-05-07	2020-05-25
2020-07-31	2020-08-10	2020-11-14	2020-12-25	2021-01-01
2021-02-12	2021-04-02	2021-05-01	2021-08-09	2021-12-25
2022-01-03	2022-01-31	2022-02-01	2022-02-02	2022-04-15
2022-05-02	2022-08-09	2022-12-26	2022-12-31	2025-01-01
2025-04-18	2025-12-25	2026-01-01		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The MSCI Taiwan Index Futures contract in the SGX Future market trades in the `Asia/Singapore` time zone.

Asset Classes

Future Options

Future Option contracts give the buyer a window of opportunity to buy or sell the underlying Future contract at a specific price.

Requesting Data

Handling Data

Greeks and Implied Volatility

Market Hours

See Also

BasicTemplateFutureOptionAlgorithm.py
BasicTemplateFutureOptionAlgorithm.cs

Future Options

Requesting Data

Request Future Options data in your algorithm to receive a feed of contract prices in the `OnData` method. For more information about the specific dataset we use for backtests, see the US Future Options dataset listing . To trade Future Options live, you can use one of the brokerage data providers . We currently only support European-style Options for US Future Options.

To request Future Options data, call one of the following methods:

- `AddFutureOption` to add a basket of contracts for a single underlying Future
- `AddFutureOptionContract` to add individual contracts for an underlying Future

Request and Filter Contracts for Option Strategies

`AddFutureOption`

Request and Filter All Contracts

`AddFutureOption`

Select Individual Contracts

`AddFutureOptionContract`

See Also

Introduction to Options
Applied Options

Requesting Data

Universes

Introduction

The `AddFutureOption` method enables you to select a basket of Option contracts for an underlying Future. To form your universe of contracts, you can filter them down by their strike price and expiry or you can choose a subset of the contracts that form popular Option strategies . If you want to subscribe to individual contracts one-by-one instead of a set of contracts, see Individual Contracts .

Create Universes

To add a universe of Future Option contracts, in the `Initialize` method, call the `AddFutureOption` method. Pass the `optionFilter` argument to filter the set of tradable contract down to just the contracts you want.

```C#
public class BasicFutureOptionAlgorithm : QCAlgorithm
{
    private Future _future;
    public override void Initialize()
    {
        _future = AddFuture(Futures.Indices.SP500EMini,
            extendedMarketHours: true,
            dataMappingMode: DataMappingMode.OpenInterest,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            contractDepthOffset: 0);
        _future.SetFilter(0, 182);

        AddFutureOption(_future.Symbol, optionFilterUniverse => optionFilterUniverse.Strikes(-1, 1));
    }

    public override void OnData(Slice data)
    {
        // Iterate all Option chains to find the desired future and future options contracts
        foreach (var (canonicalSymbol, chain) in data.OptionChains)
        {
            var futureContract = canonicalSymbol.Underlying;
            foreach (var (symbol, contract) in chain.Contracts)
            {
                var strike = contract.Strike;
            }
        }

        // Get the Option chain for the mapped contract
        if (data.OptionChains.TryGetValue(QuantConnect.Symbol.CreateCanonicalOption(_future.Mapped), out var
mappedChain))
        {
            foreach (var (symbol, contract) in mappedChain.Contracts)
            {
                var strike = contract.Strike;
            }
        }
    }
}
```

For more information about the `AddFutureOption` method, see Create Universes .

Filter by Investment Strategy

Options trading strategies consist of simultaneously buying or selling one or more Option contracts of the same underlying Future with different features. The following table describes the filter methods of the `OptionFilterUniverse` class that select

contract for Option strategies.

The following table describes the filter methods of the `OptionFilterUniverse` class that select contracts for Option strategies:

`NakedCall(int minDaysTillExpiry, decimal strikeFromAtm)` Selects a call contract to form Naked Call , Covered Call , or Protective Call Option strategies.
`NakedPut(int minDaysTillExpiry, decimal strikeFromAtm)` Selects a put contract to form Naked Put , Covered Put , or Protective Put Option strategies.
`CallSpread(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects two call contracts to form Bull Call Spread or Bear Call Spread Option strategies.
`PutSpread(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects two put contracts to form Bull Put Spread or Bear Put Spread Option strategies.
`CallCalendarSpread(decimal strikeFromAtm, int minNearDaysTillExpiry, int minFarDaysTillExpiry)` Selects two call contracts to form Long Call Calendar Spread or Short Call Calendar Spread Option strategies.
`PutCalendarSpread(decimal strikeFromAtm, int minNearDaysTillExpiry, int minFarDaysTillExpiry)` Selects two put contracts to form Long Put Calendar Spread or Short Put Calendar Spread Option strategies.
`Strangle(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects two contracts to form Long Strangle or Short Strangle Option strategies.
`Straddle(int minDaysTillExpiry)` Selects two contracts to form Long Straddle or Short Straddle Option strategies.
`ProtectiveCollar(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects two contracts to form Protective Collar Option strategies.
`Conversion(int minDaysTillExpiry, decimal strikeFromAtm)` Selects two contracts to form Conversion or Reverse Conversion Option strategies.
`CallButterfly(int minDaysTillExpiry, decimal strikeSpread)` Selects three contracts to form Long Call Butterfly or Short Call Butterfly Option strategies.

`PutButterfly(int minDaysTillExpiry, decimal strikeSpread)` Selects three contracts to form Long Put Butterfly or Short Put Butterfly Option strategies.
`IronButterfly(int minDaysTillExpiry, decimal strikeSpread)` Selects four contracts to form Long Iron Butterfly or Short Iron Butterfly Option strategies.
`IronCondor(int minDaysTillExpiry, decimal nearStrikeSpread, decimal farStrikeSpread)` Selects four contracts to form Long Iron Condor or Short Iron Condor Option strategies.
`BoxSpread(int minDaysTillExpiry, decimal strikeSpread)` Selects four contracts to form Box Spread or Short Box Spread Option strategies.
`JellyRoll(decimal strikeFromAtm, int minNearDaysTillExpiry, int minFarDaysTillExpiry)` Selects four contracts to form Jelly Roll or Short Jelly Roll Option strategies.
`CallLadder(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal middleStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects four contracts to form Bear Call Ladder or Bull Call Ladder Option strategies.
`PutLadder(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal middleStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects four contracts to form Bear Put Ladder or Bull Put Ladder Option strategies.

The preceding methods return an `OptionFilterUniverse` , so you can chain the methods together.

```C#
// Example 1: Select a Straddle
AddFutureOption(_future.Symbol, optionFilterUniverse => optionFilterUniverse.Straddle(30, 5, 10));

// Example 2: Select the contracts (including weeklys) that expire in the next 30 days that form an Iron Condor
AddFutureOption(_future.Symbol, optionFilterUniverse => optionFilterUniverse.Strikes(-20, 20).Expiration(0,
30).IronCondor(30, 5, 10));

// Example 3: Select the front month contracts that form an Strangle
AddFutureOption(_future.Symbol, optionFilterUniverse => optionFilterUniverse.FrontMonth().Strangle(30, 5, -10));
```

Filter by Other Contract Properties

To set a contract filter, in the `Initialize` method, call the `AddFutureOption` method and pass the `optionFilter` argument.

```
C#
// Select contracts that have a strike price within 1 strike level above and below the underlying price
AddFutureOption(_future.Symbol, optionFilterUniverse => optionFilterUniverse.Strikes(-1,1));

// Select contracts that expire within 30 days
AddFutureOption(_future.Symbol, optionFilterUniverse => optionFilterUniverse.Expiration(0, 30));

// Select contracts that have a strike price within 1 strike level and expire within 30 days
AddFutureOption(_future.Symbol, optionFilterUniverse => optionFilterUniverse.Strikes(-1,1).Expiration(0, 30));

// Select call contracts
AddFutureOption(_future.Symbol, optionFilterUniverse => optionFilterUniverse.CallsOnly());
```

The following table describes the filter methods of the `OptionFilterUniverse` class:

`Strikes(int minStrike, int maxStrike)` Selects contracts that are within `minStrike` strikes below the underlying price and `maxStrike` strikes above the underlying price.
`CallsOnly()` Selects call contracts.
`PutsOnly()` Selects put contracts.
`StandardsOnly()` Selects standard contracts.
`IncludeWeeklys()` Selects non-standard weeklys contracts. This filter doesn't currently work. To add this functionality to our roadmap , react to this GitHub Issue with a thumbs up emoji.
`WeeklysOnly()` Selects weekly contracts. This filter doesn't currently work. To add this functionality to our roadmap , react to this GitHub Issue with a thumbs up emoji.
`FrontMonth()` Selects the front month contract.
`BackMonths()` Selects the non-front month contracts.

`BackMonth()` Selects the back month contracts.
`Expiration(int minExpiryDays, int maxExpiryDays)` Selects contracts that expire within a range of dates relative to the current day.
`Contracts(IEnumerable<Symbol> contracts)` Selects a list of contracts.
`Contracts(Func<IEnumerable<Symbol>, IEnumerable< Symbol>> contractSelector)` Selects contracts that a selector function selects.

The preceding methods return an `OptionFilterUniverse`, so you can chain the methods together.

The preceding methods return an `OptionFilterUniverse`, so you can chain the methods together.

```C#
// Example 1: Select the front month call contracts
AddFutureOption(_future.Symbol, optionFilterUniverse => optionFilterUniverse.CallsOnly().FrontMonth());

// Example 2: Select the contracts that expire in the next 90 days
AddFutureOption(_future.Symbol, optionFilterUniverse => optionFilterUniverse.Strikes(-20, 20).Expiration(0, 90));
```

To perform thorough filtering on the `OptionFilterUniverse`, define an isolated filter method.

```C#
public class BasicFutureOptionAlgorithm : QCAlgorithm
{
    private Future _future;
    public override void Initialize()
    {
        AddFutureOption(_future.Symbol, Selector);
    }

    private OptionFilterUniverse Selector(OptionFilterUniverse optionFilterUniverse)
    {
        var symbols = optionFilterUniverse.PutsOnly();
        var strike = symbols.Select(symbol => symbol.ID.StrikePrice).Min();
        symbols = symbols.Where(symbol => symbol.ID.StrikePrice == strike);
        return optionFilterUniverse.Contracts(symbols);
    }
}
```

Some of the preceding filter methods only set an internal enumeration in the `OptionFilterUniverse` that it uses later on in the filter process. This subset of filter methods don't immediately reduce the number of contract `Symbol` objects in the `OptionFilterUniverse`.

Default Filter

By default, LEAN subscribes to the Option contracts that have the following characteristics:

- Standard type (exclude weeklys)
- Within 1 strike price of the underlying asset price
- Expire within 35 days

To adjust the universe of contracts, set a filter. The filter usually runs at the first bar of every day. When the filter selects a contract that isn't currently in your universe, LEAN adds the new contract data to the next `Slice` that it passes to the `OnData` method.

Historical Data

To get historical data for a Future Options universe, see Historical Data .

Examples

The following examples demonstrate some common practices for requesting Future Option data through universe filtering.

Example 1: Call Spread

The following example demonstrates how to implement a bull call spread option strategy using universe filtering. We have to first subscribe to the underlying Future (with filtering), then call `AddFutureOption` with the underlying `Symbol` and the filtering function. We use `CallSpread` method in the `OptionFilterUniverse` object to return only the 2 best match call contracts that forms a bull call spread.

C#

```
public class FutureOptionAlgorithm : QCAlgorithm
{
    private Future _underlying;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Filter the underlying continuous Futures to narrow the FOP spectrum.
        _underlying = AddFuture(Futures.Indices.SP500EMini,
            extendedMarketHours: true,
            dataMappingMode: DataMappingMode.OpenInterest,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            contractDepthOffset: 0);
        _underlying.SetFilter(0, 182);
        // Use CallSpread filter to obtain the 2 best-matched contracts that forms a call spread.
        // It simplifies from further filtering and reduce computation on redundant subscription.
        AddFutureOption(_underlying.Symbol, (u) => u.CallSpread(5, 5, -5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested)
            return;

        // Create canonical symbol for the mapped future contract, since we need that to access the option chain.
        var symbol = QuantConnect.Symbol.CreateCanonicalOption(_underlying.Mapped);

        // Get option chain data for the mapped future only.
        // It requires 2 contracts with different strikes to form a call spread, so we make sure at least 2 contracts
are present.
        if (!slice.OptionChains.TryGetValue(symbol, out var chain) || chain.Count() < 2)
            return;

        // Separate the contracts by strike, as we need to access their strike.
        var expiry = chain.Min(x => x.Expiry);
        var itmStrike = chain.Min(x => x.Strike);
        var otmStrike = chain.Max(x => x.Strike);

        // Use abstraction method to order a bull call spread to avoid manual error.
        var optionStrategy = OptionStrategies.BullCallSpread(symbol, itmStrike, otmStrike, expiry);
        Buy(optionStrategy, 1);
    }
}
```

Note that since both the underlying Future and the Future Option are expiring on the same day and are cash-settling in most cases, Lean can exercise the Future Option into account cash automatically at expiry and we do not need to handle the option exercise/assignment event.

Requesting Data

Individual Contracts

Introduction

The `AddFutureOptionContract` method enables you to add an individual Option contract to your algorithm. To check which contracts are currently available to add to your algorithm, use the `OptionChain` method. If you want to subscribe to a set of contracts instead of individual contracts one-by-one, see Universes .

Create Subscriptions

Before you can subscribe to an Option contract, you must configure the underlying Future and get the contract `Symbol` .

C#

```
public class BasicFutureOptionAlgorithm : QCAlgorithm
{
    private Future _future;
    private Symbol _contractSymbol;
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        _future = AddFuture(Futures.Indices.SP500EMini,
            extendedMarketHours: true,
            dataMappingMode: DataMappingMode.OpenInterest,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            contractDepthOffset: 0);
        _future.SetFilter(0, 182);
    }

    public override void OnData(Slice data)
    {
        if (_contractSymbol == null)
        {
            // Method 1: Add the desired option chain for the mapped contract
            _contractSymbol = SelectOptionContract(_future.Mapped);
            AddFutureOptionContract(_contractSymbol);

            // Method 2: Get all future contracts from the Future chain provider
            var futureContractSymbol = FuturesChain(_future.Symbol).First().Symbol;
            var symbol = SelectOptionContract(futureContractSymbol);
            AddFutureOptionContract(symbol);

            // Method 3: Iterate all Future chains to find the desired future options contracts
            foreach (var (_, chain) in data.FutureChains)
            {
                var expiry = chain.Min(contract => contract.Expiry);
                var futureContract = chain.Where(contract => contract.Expiry == expiry).FirstOrDefault();
                symbol = SelectOptionContract(futureContract.Symbol);
                AddFutureOptionContract(symbol);
            }
        }
    }

    private Symbol SelectOptionContract(Symbol futureContractSymbol)
    {
        var chain = OptionChain(futureContractSymbol)
            .Where(contract => contract.Right == OptionRight.Call).ToList();
        var expiry = chain.Min(contract => contract.Expiry);
        return chain
            .Where(contract => contract.Expiry == expiry)
            .OrderBy(contract => contract.Strike)
            .Select(contract => contract.Symbol).FirstOrDefault();
    }
}
```

Configure the Underlying Futures Contracts

In most cases, you should subscribe to the underlying Futures contract before you subscribe to a Futures Option contract.

```
C#

_future = AddFuture(Futures.Indices.SP500EMini,
    extendedMarketHours: true,
    dataMappingMode: DataMappingMode.OpenInterest,
    dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
    contractDepthOffset: 0);
```

To get a `Symbol` of a specific Futures contract, use the `Mapped` property of the `Future` object or the `Symbol` property of the `FutureContract` objects in the `Slice.FutureChains` collection.

Get Contract Symbols

To subscribe to a Future Option contract, you need the contract `Symbol` . The preferred method to getting Option contract `Symbol` objects is to use the `OptionChain` method. This method returns an `OptionChain` object, which represent an entire chain of Option contracts for a single underlying security. With the chain, sort and filter the data to find the specific contract(s) you want to trade.

```
C#

// Get the contracts available to trade.
var chain = OptionChain(futureContractSymbol)
    .Where(contract => contract.Right == OptionRight.Call).ToList();

// Select a contract.
var expiry = chain.Min(contract => contract.Expiry);
_contractSymbol = chain
    // Select call contracts with the closest expiry.
    .Where(contract => contract.Expiry == expiry)
    // Select the contract with the lowest strike price.
    .OrderBy(contract => contract.Strike)
    .Select(contract => contract.Symbol).FirstOrDefault();
```

Alternatively, you can create a Future Option contract `Symbol` object directly parsing its OSI ticker using the `SymbolRepresentation.ParseOptionTickerOSI` method.

```
C#

// Create the contract Symbol for the Sept 2024 E-mini S&P 500 Call 7900 20 Sept 2024.
_contractSymbol = SymbolRepresentation.ParseOptionTickerOSI("ESU24 240920C07900000", SecurityType.FutureOption,
OptionStyle.American, Market.CME);
// Create the contract Symbol using the 1-digit year format.
_contractSymbol = SymbolRepresentation.ParseOptionTickerOSI("ESU4 240920C07900000", SecurityType.FutureOption,
OptionStyle.American, Market.CME);
```

Subscribe to Contracts

To create a Future Option contract subscription, pass the contract `Symbol` to the `AddFutureOptionContract` method. Save a reference to the contract `Symbol` so you can easily access the Option contract in the OptionChain that LEAN passes to the `OnData` method. This method returns an `Option` object. To override the default pricing model of the Option, set a pricing model .

```
C#

var option = AddFutureOptionContract(_contractSymbol);
option.PriceModel = OptionPriceModels.BinomialCoxRossRubinstein();
```

Warm Up Contract Prices

If you subscribe to a Future Option contract with `AddFutureOptionContract` , you'll need to wait until the next `Slice` to receive

data and trade the contract. To trade the contract in the same time step you subscribe to the contract, set the current price of the contract in a security initializer .

```
C#
Settings.SeedInitialPrices = true;
```

Supported Assets

To view the supported assets in the US Future Options dataset, see the Supported Assets .

Resolutions

The following table shows the available resolutions and data formats for Future Option contract subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`				
`Second`				
`Minute`	✓	✓		
`Hour`	✓	✓		
`Daily`	✓	✓		

There is only one resolution option, so you don't need to pass a `resolution` argument to the `AddFutureOptionContract` method.

```
C#
AddFutureOptionContract(_optionContractSymbol, Resolution.Minute);
```

To create custom resolution periods, see Consolidating Data .

Supported Markets

The following `Market` enumeration members are available for Future Options:

You don't need to pass a `Market` argument to the `AddFutureOptionContract` method because the contract `Symbol` already contains the market.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
C#
AddFutureOptionContract(_optionContractSymbol, fillForward: false);
```

Margin and Leverage

LEAN models buying power and margin calls to ensure your algorithm stays within the margin requirements. Future Options are already leveraged products, so you can't change their leverage.

Extended Market Hours

By default, your security subscriptions only cover regular trading hours. To subscribe to pre and post-market trading hours for a specific asset, enable the `extendedMarketHours` argument when you create the security subscription.

```
C#
AddFutureOptionContract(_optionContractSymbol, extendedMarketHours: true);
```

You only receive extended market hours data if you create the subscription with minute, second, or tick resolution. If you create the subscription with daily or hourly resolution, the bars only reflect the regular trading hours.

To view the schedule of regular and extended market hours, see Market Hours .

Data Normalization

The data normalization mode doesn't affect the data that LEAN passes to `OnData` or the data from history request . If you change the data normalization mode, it won't change the outcome.

Remove Subscriptions

To remove a contract subscription that you created with `AddFutureOptionContract` , call the `RemoveOptionContract` method. This method is an alias for `RemoveSecurity` .

```
C#
RemoveOptionContract(_optionContractSymbol);
```

The `RemoveOptionContract` method cancels your open orders for the contract and liquidates your holdings.

Properties

The `AddFutureOptionContract` method returns an `Option` object, which have the following properties:

Helper Methods

The `Option` object provides methods you can use for basic calculations. These methods require the underlying price. To get the `Option` object and the `Security` object for its underlying in any function, use the Option `Symbol` to access the value in the `Securities` object.

```
C#
var option = Securities[_optionContractSymbol];
var underlying = Securities[_optionContractSymbol.Underlying];
var underlyingPrice = underlying.Price;
```

To get the Option payoff , call the `GetPayOff` method.

```
C#
var payOff = option.GetPayOff(underlyingPrice);
```

To get the Option intrinsic value , call the `GetIntrinsicValue` method.

```
C#
var intrinsicValue = option.GetIntrinsicValue(underlyingPrice);
```

To get the Option out-of-the-money amount , call the `OutOfTheMoneyAmount` method.

```C#
var otmAmount = option.OutOfTheMoneyAmount(underlyingPrice);
```

To check whether the Option can be automatic exercised, call the `IsAutoExercised` method.

```C#
var isAutoExercised = option.IsAutoExercised(underlyingPrice);
```

Future Options

Handling Data

Introduction

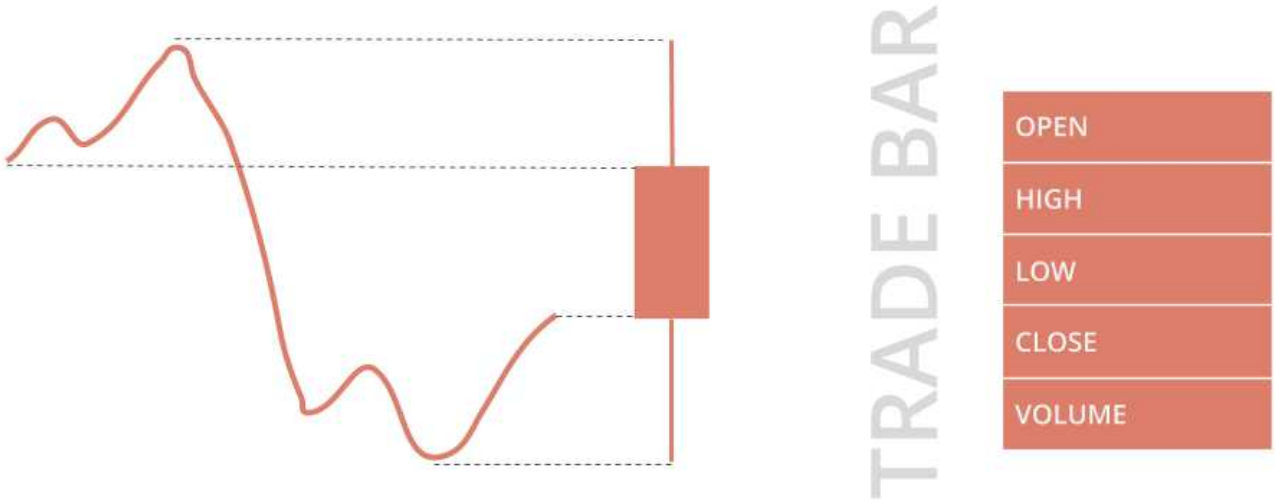
LEAN passes the data you request to the `OnData` method so you can make trading decisions. The default `OnData` method accepts a `Slice` object, but you can define additional `OnData` methods that accept different data types. For example, if you define an `OnData` method that accepts a `TradeBar` argument, it only receives `TradeBar` objects. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `TradeBars` `DataDictionary` is made up of `TradeBar` objects. To access individual data points in the dictionary, you can index the dictionary with the Futures Option contract ticker or `Symbol` , but we recommend you use the `Symbol` .

To view the resolutions that are available for Future Options data, see Resolutions .

Trades

`TradeBar` objects are price bars that consolidate individual trades from the exchanges. They contain the open, high, low, close, and volume of trading activity over a period of time.



To get the `TradeBar` objects in the `Slice` , index the `Slice` or index the `Bars` property of the `Slice` with the Option contract `Symbol` . If the Option contract doesn't actively trade or you are in the same time step as when you added the Option contract subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your Option contract before you index the `Slice` with the Option contract `Symbol` .

```C#
public override void OnData(Slice slice)
{
    // Check if the symbol is contained in TradeBars object
    if (slice.Bars.ContainsKey(_optionContractSymbol))
    {
        // Obtain the mapped TradeBar of the symbol
        var tradeBar = slice.Bars[_optionContractSymbol];
    }
}
```

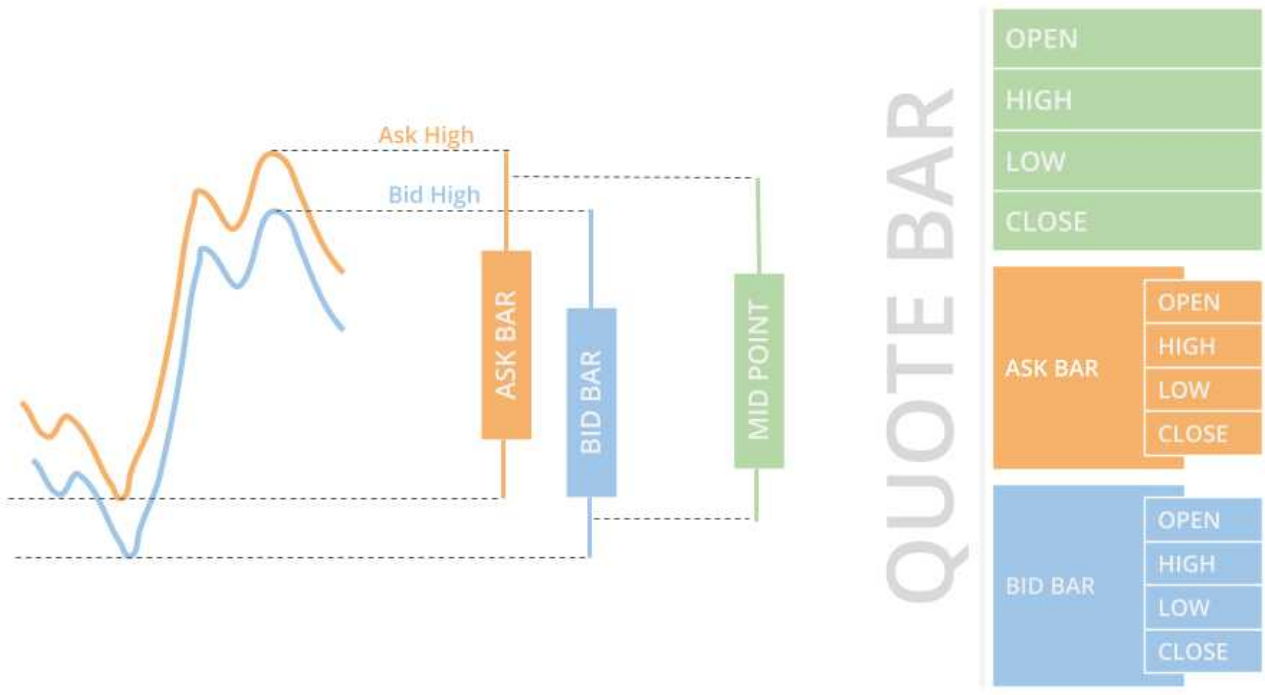
You can also iterate through the `TradeBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `TradeBar` objects.

```C#
public override void OnData(Slice slice)
{
    // Iterate all received Symbol-TradeBar key-value pairs
    foreach (var kvp in slice.Bars)
    {
        var symbol = kvp.Key;
        var tradeBar = kvp.Value;
        var closePrice = tradeBar.Close;
    }
}
```

`TradeBar` objects have the following properties:

Quotes

`QuoteBar` objects are bars that consolidate NBBO quotes from the exchanges. They contain the open, high, low, and close prices of the bid and ask. The `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` object are the mean of the respective bid and ask prices. If the bid or ask portion of the `QuoteBar` has no data, the `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` copy the values of either the `Bid` or `Ask` instead of taking their mean.



To get the `QuoteBar` objects in the `Slice` , index the `QuoteBars` property of the `Slice` with the Option contract `Symbol` . If the Option contract doesn't actively get quotes or you are in the same time step as when you added the Option contract

subscription, the `Slice` may not contain data for your `Symbol`. To avoid issues, check if the `Slice` contains data for your Option contract before you index the `Slice` with the Option contract `Symbol`.

```
C#
public override void OnData(Slice slice)
{
    // Check if the symbol is contained in QuoteBars object
    if (slice.QuoteBars.ContainsKey(_optionContractSymbol))
    {
        // Obtain the mapped QuoteBar of the symbol
        var quoteBar = slice.QuoteBars[_optionContractSymbol];
    }
}
```

You can also iterate through the `QuoteBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `QuoteBar` objects.

```
C#
public override void OnData(Slice slice)
{
    // Iterate all received Symbol-QuoteBar key-value pairs
    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        var askPrice = quoteBar.Ask.Close;
    }
}
```

`QuoteBar` objects let LEAN incorporate spread costs into your simulated trade fills to make backtest results more realistic.

`QuoteBar` objects have the following properties:

Futures Chains

`FuturesChain` objects represent an entire chain of contracts for a single underlying Future. To get the `FuturesChain`, index the `FuturesChains` property of the `Slice` with the continuous contract `Symbol`.

```
C#
public override void OnData(Slice slice)
{
    // Try to get the FutureChain using the canonical symbol
    if (slice.FuturesChains.TryGetValue(_futureContractSymbol.Canonical, out var chain))
    {
        // Get all contracts if the FutureChain contains any member
        var contracts = chain.Contracts;
    }
}
```

You can also loop through the `FuturesChains` property to get each `FuturesChain`.

```C#
public override void OnData(Slice slice)
{
    // Iterate all received Canonical Symbol-FutureChain key-value pairs
    foreach (var kvp in slice.FuturesChains)
    {
        var continuousContractSymbol = kvp.Key;
        var chain = kvp.Value;
        var contracts = chain.Contracts;
    }
}

// Using this overload will only handle any FutureChains object received
public void OnData(FuturesChains futuresChains)
{
    // Iterate all received Canonical Symbol-FutureChain key-value pairs
    foreach (var kvp in futuresChains)
    {
        var continuousContractSymbol = kvp.Key;
        var chain = kvp.Value;
        var contracts = chain.Contracts;
    }
}
```

`FuturesChain` objects have the following properties:

Futures Contracts

`FuturesContract` objects represent the data of a single Futures contract in the market. To get the Futures contracts in the `Slice`, use the `Contracts` property of the `FuturesChain`.

```C#
public override void OnData(Slice slice)
{
    // Try to get the FutureChain using the canonical symbol
    if (slice.FuturesChains.TryGetValue(_futureContractSymbol.Canonical, out var chain))
    {
        // Get individual contract data
        if (chain.Contracts.TryGetValue(_futureContractSymbol, out var contract))
        {
            var price = contract.LastPrice;
        }
    }
}

// // Using this overload will only handle any FutureChains object received
public void OnData(FuturesChains futuresChains)
{
    // Try to get the FutureChain using the canonical symbol
    if (futuresChains.TryGetValue(_futureContractSymbol.Canonical, out var chain))
    {
        // Get individual contract data
        if (chain.Contracts.TryGetValue(_futureContractSymbol, out var contract))
        {
            var price = contract.LastPrice;
        }
    }
}
```

`FuturesContract` objects have the following properties:

Option Chains

`OptionChain` objects represent an entire chain of Option contracts for a single underlying security.

To get the `OptionChain`, index the `OptionChains` property of the `Slice` with the canonical `Symbol`.

```
C#
public override void OnData(Slice slice)
{
    // Try to get the OptionChain using the canonical symbol
    if (slice.OptionChains.TryGetValue(_optionContractSymbol.Canonical, out var chain))
    {
        // Get all contracts if the OptionChain contains any member
        var contracts = chain.Contracts;
    }
}
```

You can also loop through the `OptionChains` property to get each `OptionChain` .

```
C#
public override void OnData(Slice slice)
{
    // Iterate all received Canonical Symbol-OptionChain key-value pairs
    foreach (var kvp in slice.OptionChains)
    {
        var canonicalFOPSymbol = kvp.Key;
        var chain = kvp.Value;
        var contracts = chain.Contracts;
    }
}
```

`OptionChain` objects have the following properties:

Option Contracts

`OptionContract` objects represent the data of a single Option contract in the market. To get the Option contracts in the `Slice` , use the `Contracts` property of the `OptionChain` .

```
C#
public override void OnData(Slice slice)
{
    // Try to get the OptionChain using the canonical symbol
    if (slice.OptionChains.TryGetValue(_optionContractSymbol.Canonical, out var chain))
    {
        // Get individual contract data
        if (chain.Contracts.TryGetValue(_optionContractSymbol, out var contract))
        {
            var price = contract.Price;
        }
    }
}
```

You can also iterate through the `FuturesChains` first.

```C#
public override void OnData(Slice slice)
{
    // Iterate all received Canonical Symbol-FuturesChain key-value pairs
    foreach (var kvp in slice.FuturesChains)
    {
        var continuousContractSymbol = kvp.Key;
        var futuresChain = kvp.Value;

        // Select a Future Contract and create its canonical FOP Symbol
        var futuresContract = futuresChain.First();
        var canonicalFOPSymbol = QuantConnect.Symbol.CreateCanonicalOption(futuresContract.Symbol);
        // Try to get the OptionChain using the canonical FOP symbol
        if (slice.OptionChains.TryGetValue(canonicalFOPSymbol, out var optionChain))
        {
            // Get individual contract data
            if (optionChain.Contracts.TryGetValue(_optionContractSymbol, out var optionContract))
            {
                var price = optionContract.Price;
            }
        }
    }
}
```

Greeks and Implied Volatility

To get the Greeks and implied volatility of an Option contract, use the `Greeks` and `implied_volatility` members.

```C#
public override void OnData(Slice slice)
{
    // Try to get the OptionChain using the canonical symbol
    if (slice.OptionChains.TryGetValue(_optionContractSymbol.Canonical, out var chain))
    {
        // Get individual contract data
        if (chain.Contracts.TryGetValue(_optionContractSymbol, out var contract))
        {
            // Get greeks data of the selected contract
            var delta = contract.Greeks.Delta;
            var iv = contract.ImpliedVolatility;
        }
    }
}
```

LEAN only calculates Greeks and implied volatility when you request them because they are expensive operations. If you invoke the `Greeks` property, the Greeks aren't calculated. However, if you invoke the `Greeks.Delta` , LEAN calculates the delta. To avoid unecessary computation in your algorithm, only request the Greeks and implied volatility when you need them. For more information about the Greeks and implied volatility, see Options Pricing .

Open Interest

Open interest is the number of outstanding contracts that haven't been settled. It provides a measure of investor interest and the market liquidity, so it's a popular metric to use for contract selection. Open interest is calculated once per day. To get the latest open interest value, use the `OpenInterest` property of the `Option` or `OptionContract` .

```C#
public override void OnData(Slice slice)
{
    // Try to get the OptionChains using the canonical symbol
    if (slice.OptionChains.TryGetValue(_contractSymbol.Canonical, out var chain))
    {
        // Get individual contract data
        if (chain.Contracts.TryGetValue(_contractSymbol, out var contract))
        {
            // Get the open interest of the selected contracts
            var openInterest = contract.OpenInterest;
        }
    }
}

public void OnData(OptionChains optionChains)
{
    // Try to get the OptionChains using the canonical symbol
    if (optionChains.TryGetValue(_contractSymbol.Canonical, out var chain))
    {
        // Get individual contract data
        if (chain.Contracts.TryGetValue(_contractSymbol, out var contract))
        {
            // Get the open interest of the selected contracts
            var openInterest = contract.OpenInterest;
        }
    }
}
```

Properties

`OptionContract` objects have the following properties:

Examples

The following examples demonstrate some common practices for handling Future Option data.

Example 1: Monthly Protective Put

The following algorithm shows how to perform monthly selection on individual ES Future Option contract to implement a protective put option strategy to hedge speculation on S&P500 Future. It is a useful tool to hedge the excessive risk on leverage using Futures to trade.

C#

```
public class FutureOptionExampleAlgorithm : QCAlgorithm
{
    private Future _underlying;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000000);

        // Seed the security price to ensure the underlying price is available at the initial filtering.
        Settings.SeedInitialPrices = true;
        // Set data normalization mode to raw for fair strike price comparison.
        UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;
        // Subscribe the underlying since the updated price is needed for filtering.
        _underlying = AddFuture(Futures.Indices.SP500EMini,
            dataMappingMode: DataMappingMode.OpenInterest,
            dataNormalizationMode: DataNormalizationMode.Raw,
            contractDepthOffset: 0);
        // Filter the underlying continuous Futures to narrow the FOP spectrum.
        _underlying.SetFilter(0, 31);

        // Schedule a monthly event on selection of future-future option pair, since the portfolio rebalance on a
monthly basis.
        Schedule.On(
            DateRules.MonthStart(_underlying.Symbol),
            TimeRules.AfterMarketOpen(_underlying.Symbol, 0),
            SelectionAndRebalance
        );
    }

    private void SelectionAndRebalance()
    {
        // Get all available put FOP contract for the mapped underlying contract, since the trade liquidity and
volatility is the highest.
        var contractSymbols = OptionChain(_underlying.Mapped)
            .Where(symbol => symbol.ID.OptionRight == OptionRight.Put)
            .ToList();
        // Select the ATM put expires the same date as the underlying. The max expiry of the FOP will expire the same
time as the front month future.
        var expiry = contractSymbols.Max(symbol => symbol.ID.Date);
        var selected = contractSymbols.Where(symbol => symbol.ID.Date == expiry)
            .OrderBy(symbol => Math.Abs(symbol.ID.StrikePrice - Securities[_underlying.Mapped].Price))
            .First();
        // Request the FOP contract data for trading.
        var contract = AddFutureOptionContract(selected);

        // A Protective Put consists of long a lot of the underlying, and long a put contract.
        MarketOrder(_underlying.Mapped, contract.SymbolProperties.ContractMultiplier);
        MarketOrder(contract.Symbol, 1);
    }
}
```

Example 2: Weekly Covered Call

The below example demonstrates a weekly-renewing covered call strategy to collect credit of selling the option. It filters the ATM call contract that expires within the current week at week start using `SetFilter` filtering function.

C#

```
public class FutureOptionExampleAlgorithm : QCAlgorithm
{
    private Future _underlying;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000000);
        // Set data normalization mode to raw for fair strike price comparison.
        UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;

        // Subscribe the underlying since the updated price is needed for filtering.
        _underlying = AddFuture(Futures.Indices.SP500EMini,
            extendedMarketHours: true,
            dataMappingMode: DataMappingMode.OpenInterest,
            dataNormalizationMode: DataNormalizationMode.Raw,
            contractDepthOffset: 0);
        // Filter the underlying continuous Futures to narrow the FOP spectrum.
        _underlying.SetFilter(0, 182);
        // Filter for the current-week-expiring calls to formulate a covered call that expires at the end of week.
        AddFutureOption(_underlying.Symbol, (u) => u.IncludeWeeklys().CallsOnly().Expiration(0, 5));
    }

    public override void OnData(Slice slice)
    {
        // Create canonical symbol for the mapped future contract, since option chains are mapped by canonical
symbol.
        var symbol = QuantConnect.Symbol.CreateCanonicalOption(_underlying.Mapped);

        // Get option chain data for the mapped future, as both the underlying and FOP have the highest liquidity
among all other contracts.
        if (!Portfolio.Invested &&
            slice.OptionChains.TryGetValue(symbol, out var chain))
        {
            // Obtain the ATM call that expires at the end of week, such that both underlying and the FOP expires the
same time.
            var expiry = chain.Max(x => x.Expiry);
            var atmCall = chain.Where(x => x.Expiry == expiry)
                .OrderBy(x => Math.Abs(x.Strike - x.UnderlyingLastPrice))
                .First();

            // Use abstraction method to order a covered call to avoid manual error.
            var optionStrategy = OptionStrategies.CoveredCall(symbol, atmCall.Strike, expiry);
            Buy(optionStrategy, 1);
        }
    }
}
```

Note that since both the underlying Future and the Future Option are expiring on the same day and are cash-settling in most cases, Lean can exercise the Future Option into account cash automatically at expiry and we do not need to handle the option exercise/assignment event.

Future Options

Greeks and Implied Volatility

The Greeks are measures that describe the sensitivity of an Option's price to various factors like underlying price changes (Delta), time decay (Theta), volatility (Vega), and interest rates (Rho), while Implied Volatility (IV) represents the market's expectation of the underlying asset's volatility over the life of the Option.

Key Concepts

Concepts on Greeks, IV, and IV modeling

Indicators

Implementation of IV and Greeks as LEAN indicators

See Also

Option Pricing
Risk Free Interest Rate
Dividend Yield

Greeks and Implied Volatility

Key Concepts

Introduction

LEAN provides multiple ways for you to get the Greeks and implied volatility (IV) of Option contracts. There is a daily, pre-calculated dataset based on the end of the previous trading day. There is also a stream of values that are calculated on-the-fly as your algorithm runs.

Greeks on Universes

This feature is coming soon for Future Options.

Greek History

This feature is coming soon for Future Options.

Greeks on Data Events

The Greeks and IV values that you get from the current Slice are from the price model calculations. These values aren't the same as the values you get in the universe filter function.

C#

```
public class FutureOptionDataEventsAlgorithm : QCAlgorithm
{
    private Future _future;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        Settings.SeedInitialPrices = true;
        AddSecurityInitializer(security =>
        {
            if (security.Type == SecurityType.FutureOption)
            {
                (security as Option).PriceModel = OptionPriceModels.CrankNicolsonFD();
            }
        });
        _future = AddFuture(Futures.Indices.SP500EMini,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            dataMappingMode: DataMappingMode.OpenInterest,
            contractDepthOffset: 0);
        _future.SetFilter(0, 182);
        AddFutureOption(_future.Symbol, universe => universe.Strikes(-5, 5));
    }

    public override void OnData(Slice data)
    {
        foreach (var chain in data.OptionChains.Values)
        {
            foreach (var contract in chain)
            {
                var iv = contract.ImpliedVolatility;
                var delta = contract.Greeks.Delta;
            }
        }
    }
}
```

To view all the models LEAN supports, see Supported Models .

Greek Indicators

Option indicators provide daily and intraday values for the Greeks and implied volatility of Future Option contracts.

Examples

For detailed examples on using Option Greeks and Implied Volatility indicators in algorithm, you may refer to the Greeks Indicators page examples.

Greeks and Implied Volatility

Indicators

Introduction

Option indicators enable you to calculate daily and intraday values for the Greeks and implied volatility of Option contracts. This page explains the most popular Option indicators and how to use them with LEAN.

Parameters

The following table describes the arguments that the automatic Option indicators methods accept:

Argument	Data Type	Description	Default Value
`symbol`	`Symbol`	The contract to use when calculating the indicator values.	
`mirrorOption`	`Symbol`	The mirror contract to use in parity type calculations.	`null`
`riskFreeRate`	`decimal`	The risk-free interest rate. If you don't provide a value, the default value is the US primary credit rate from the Interest Rate Provider Model .	`null`
`dividendYield`	`decimal`	The dividend yield rate. If you don't provide a value, the default value comes from the Dividend Yield Provider Model .	`null`
`optionModel`	`OptionPricingModelType`	The Option pricing model that's used to calculate the Greeks. If you don't provide a value, the default value is `OptionPricingModelType.BlackScholes` for European-style Options.	`null`
`resolution`	`Resolution`	The resolution of the indicator data. If you don't provide a value, the default value is the resolution of the subscription you have for the Option contract(s).	`null`

To perform implied volatility (IV) smoothing with a put-call pair, pass one of the contracts as the `symbol` argument and pass the other contract in the pair as the `mirrorOption` argument. The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, call the `SetSmoothingFunction` method of the `ImpliedVolatility` class/property.

Several different Option pricing models are supported to calculate the IV and Greeks. The following table describes the

`OptionPricingModelType` enumeration members:

Implied Volatility

Implied volatility, σ , is the market's expectation for the future volatility of an asset and is implied by the price of the assets's Options contracts. You can't observe it in the market but you can derive it from the price of an Option. For more information about implied volatility, see Implied Volatility .

Automatic Indicators

To create an automatic indicator for implied volatility, call the `QCAlgorithm.IV` method with the Option contract `Symbol` object(s).

```
C#
public class AutomaticImpliedVolatilityIndicatorAlgorithm : QCAlgorithm
{
    private Future _future;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _future = AddFuture(Futures.Indices.SP500EMini,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            dataMappingMode: DataMappingMode.OpenInterest,
            contractDepthOffset: 0);
        _future.SetFilter(0, 182);

        // Set up a Scheduled Event to select contracts and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_future.Symbol),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_future.Mapped == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_future.Mapped);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count() > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddFutureOptionContract(contracts[0]) as dynamic;
                var contract2 = AddFutureOptionContract(contracts[1]) as dynamic;

                // Create and save the automatic ImpliedVolatility indicators.
                contract1.IV = IV(contracts[0], contracts[1]);
                contract2.IV = IV(contracts[1], contracts[0]);
```

```
                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        // Get the ImpliedVolatility indicator of each contract.
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var symbol in chain)
            {
                var option = Securities[symbol] as dynamic;
                var indicator = option.IV;
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_future.Mapped].Invested)
        {
            Liquidate(_future.Mapped);
        }
    }
}
```

For more information about the `IV` method, see Parameters and Using IV Indicator .

Manual Indicators

To create a manual indicator for implied volatility, call the `ImpliedVolatility` constructor.

C#

```
public class ManualImpliedVolatilityIndicatorAlgorithm : QCAlgorithm
{
    private Future _future;
    private ConstantDividendYieldModel _dividendYieldProvider;
    // Define the Option pricing model.
    private readonly OptionPricingModelType _optionPricingModel = OptionPricingModelType.ForwardTree;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _future = AddFuture(Futures.Indices.SP500EMini,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            dataMappingMode: DataMappingMode.OpenInterest,
            contractDepthOffset: 0);
        _future.SetFilter(0, 182);

        // Set up dividend yield provider for the underlying
        _dividendYieldProvider = new ConstantDividendYieldModel(0m);

        // Set up a Scheduled Event to select contract and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_future.Symbol),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_future.Mapped == null)
        {
            return;
```

```
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_future.Mapped);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddFutureOptionContract(contracts[0]);
                var contract2 = AddFutureOptionContract(contracts[1]);

                // Create and save the manual ImpliedVolatility indicators.
                foreach (var (contractA, contractB) in new[] { (contract1, contract2), (contract2, contract1) })
                {
                    var option = contractA as dynamic;
                    option.ImpliedVolatility = new ImpliedVolatility(
                            contractA.Symbol, RiskFreeInterestRateModel, _dividendYieldProvider,
                            contractB.Symbol, _optionPricingModel
                        );
                }

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var option in chain)
            {
                var indicator = (Securities[option] as dynamic).ImpliedVolatility;
                var underlying = option.Symbol.Underlying;
                var mirror = QuantConnect.Symbol.CreateOption(
                    underlying.Value, option.ID.Market, option.ID.OptionStyle,
                    option.ID.OptionRight == OptionRight.Call ? OptionRight.Put : OptionRight.Call,
                    option.ID.StrikePrice, option.ID.Date
                ).Value;

                // Check if price data is available for both contracts and the underlying asset.
                var q = slice.QuoteBars;
                var b = slice.Bars;
                if (q.ContainsKey(option) && q.ContainsKey(mirror) && b.ContainsKey(underlying))
                {
                    var dataPoints = new List<IndicatorDataPoint>
                    {
                        new IndicatorDataPoint(option, q[option].EndTime, q[option].Close),
                        new IndicatorDataPoint(mirror, q[mirror].EndTime, q[mirror].Close),
                        new IndicatorDataPoint(
                            underlying, b[underlying].EndTime, b[underlying].Close
                        )
                    };
                    foreach (var dataPoint in dataPoints)
                    {
                        indicator.Update(dataPoint);
                    }

                    // Get the current value of the ImpliedVolatility indicator.
                    var value = indicator.Current.Value;
                }
            }
        }

        // Sell straddle as an example to trade.
```

```
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_future.Mapped].Invested)
        {
            Liquidate(_future.Mapped);
        }
    }
}
```

For more information about the `ImpliedVolatility` constructor, see Using IV Indicator .

Volatility Smoothing

The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, pass a `mirrorOption` argument to the `IV` method or `ImpliedVolatility` constructor and then call the `SetSmoothingFunction` method of the resulting `ImpliedVolatility` object. The follow table describes the arguments of the custom function:

Argument	Data Type	Description
`iv`	`decimal`	The IV of the Option contract.
`mirrorIv`	`decimal`	The IV of the mirror Option contract.

The method must return a `decimal` as the smoothened IV.

C#

```
private ImpliedVolatility _iv;

public override void Initialize()
{
    var option = QuantConnect.Symbol.CreateOption("AAPL", Market.USA, OptionStyle.American, OptionRight.Put, 505m,
new DateTime(2014, 6, 27));
    AddOptionContract(option);

    var mirrorOption = QuantConnect.Symbol.CreateOption("AAPL", Market.USA, OptionStyle.American, OptionRight.Call,
505m, new DateTime(2014, 6, 27));
    AddOptionContract(mirrorOption);

    _iv = IV(option, mirrorOption);
    // example: take average of the call-put pair
    _iv.SetSmoothingFunction((iv, mirrorIv) => (iv + mirrorIv) * 0.5m);
}
```

Delta

Delta, Δ , is the rate of change of the Option price with respect to the price of the underlying asset. It measures the first-order sensitivity of the price to a movement in underlying price. For example, an Option delta of 0.4 means that if the underlying asset moves by 1%, then the value of the Option moves by $0.4 \times 1\% = 0.4\%$. For more information about delta, see Delta .

Automatic Indicators

To create an automatic indicator for delta, call the `QCAlgorithm.D` method with the Option contract `Symbol` object(s).

C#

```
public class AutomaticDeltaIndicatorAlgorithm : QCAlgorithm
{
    private Future _future;
    private (Symbol option1, Symbol option2) _options;
```

```
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _future = AddFuture(Futures.Indices.SP500EMini,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            dataMappingMode: DataMappingMode.OpenInterest,
            contractDepthOffset: 0);
        _future.SetFilter(0, 182);

        // Set up a Scheduled Event to select contracts and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_future.Symbol),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_future.Mapped == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_future.Mapped);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count() > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddFutureOptionContract(contracts[0]) as dynamic;
                var contract2 = AddFutureOptionContract(contracts[1]) as dynamic;

                // Create and save the automatic Delta indicators.
                contract1.D = D(contracts[0], contracts[1]);
                contract2.D = D(contracts[1], contracts[0]);

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        // Get the Delta indicator of each contract.
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var symbol in chain)
            {
                var option = Securities[symbol] as dynamic;
                var indicator = option.D;
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
```

```
        // Liquidate any assigned positions.
        if (Portfolio[_future.Mapped].Invested)
        {
            Liquidate(_future.Mapped);
        }
    }
}
```

The follow table describes the arguments that the `D` method accepts in addition to the standard parameters :

Argument	Data Type	Description	Default Value
`ivModel`	`OptionPricingModelType`	The Option pricing model to use to estimate the IV when calculating Delta. If you don't provide a value, the default value is to match the `optionModel` parameter.	`null`

For more information about the `D` method, see Using D Indicator .

Manual Indicators

To create a manual indicator for delta, call the `Delta` constructor.

C#

```
public class ManualDeltaIndicatorAlgorithm : QCAlgorithm
{
    private Future _future;
    private ConstantDividendYieldModel _dividendYieldProvider;
    // Define the Option pricing model.
    private readonly OptionPricingModelType _optionPricingModel = OptionPricingModelType.ForwardTree;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _future = AddFuture(Futures.Indices.SP500EMini,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            dataMappingMode: DataMappingMode.OpenInterest,
            contractDepthOffset: 0);
        _future.SetFilter(0, 182);

        // Set up dividend yield provider for the underlying
        _dividendYieldProvider = new ConstantDividendYieldModel(0m);

        // Set up a Scheduled Event to select contract and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_future.Symbol),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_future.Mapped == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_future.Mapped);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
```

```
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddFutureOptionContract(contracts[0]);
                var contract2 = AddFutureOptionContract(contracts[1]);

                // Create and save the manual Delta indicators.
                foreach (var (contractA, contractB) in new[] { (contract1, contract2), (contract2, contract1) })
                {
                    var option = contractA as dynamic;
                    option.Delta = new Delta(
                            contractA.Symbol, RiskFreeInterestRateModel, _dividendYieldProvider,
                            contractB.Symbol, _optionPricingModel
                        );
                }

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var option in chain)
            {
                var indicator = (Securities[option] as dynamic).Delta;
                var underlying = option.Symbol.Underlying;
                var mirror = QuantConnect.Symbol.CreateOption(
                    underlying.Value, option.ID.Market, option.ID.OptionStyle,
                    option.ID.OptionRight == OptionRight.Call ? OptionRight.Put : OptionRight.Call,
                    option.ID.StrikePrice, option.ID.Date
                ).Value;

                // Check if price data is available for both contracts and the underlying asset.
                var q = slice.QuoteBars;
                var b = slice.Bars;
                if (q.ContainsKey(option) && q.ContainsKey(mirror) && b.ContainsKey(underlying))
                {
                    var dataPoints = new List<IndicatorDataPoint>
                    {
                        new IndicatorDataPoint(option, q[option].EndTime, q[option].Close),
                        new IndicatorDataPoint(mirror, q[mirror].EndTime, q[mirror].Close),
                        new IndicatorDataPoint(
                            underlying, b[underlying].EndTime, b[underlying].Close
                        )
                    };
                    foreach (var dataPoint in dataPoints)
                    {
                        indicator.Update(dataPoint);
                    }

                    // Get the current value of the Delta indicator.
                    var value = indicator.Current.Value;
                }
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_future.Mapped].Invested)
```

```
        {
            Liquidate(_future.Mapped);
        }
    }
}
```

For more information about the `Delta` constructor, see Using D Indicator .

Volatility Smoothing

The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, pass a `mirrorOption` argument to the indicator method or constructor and then call the `SetSmoothingFunction` method of the `ImpliedVolatility` property of the indicator.

C#

```
// Example: Average IV of the call-put pair.
deltaIndicator.ImpliedVolatility.SetSmoothingFunction((iv, mirrorIv) => (iv + mirrorIv) * 0.5m);
```

Gamma

Gamma, Γ , is the rate of change of the portfolio's delta with respect to the underlying asset's price. It represents the second-order sensitivity of the Option to a movement in the underlying asset's price. For more information about Gamma, see Gamma .

Automatic Indicators

To create an automatic indicator for gamma, call the `QCAlgorithm.G` method with the Option contract `Symbol` object(s).

C#

```
public class AutomaticGammaIndicatorAlgorithm : QCAlgorithm
{
    private Future _future;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _future = AddFuture(Futures.Indices.SP500EMini,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            dataMappingMode: DataMappingMode.OpenInterest,
            contractDepthOffset: 0);
        _future.SetFilter(0, 182);

        // Set up a Scheduled Event to select contracts and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_future.Symbol),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_future.Mapped == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_future.Mapped);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
```

```
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count() > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddFutureOptionContract(contracts[0]) as dynamic;
                var contract2 = AddFutureOptionContract(contracts[1]) as dynamic;

                // Create and save the automatic Gamma indicators.
                contract1.G = G(contracts[0], contracts[1]);
                contract2.G = G(contracts[1], contracts[0]);

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        // Get the Gamma indicator of each contract.
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var symbol in chain)
            {
                var option = Securities[symbol] as dynamic;
                var indicator = option.G;
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_future.Mapped].Invested)
        {
            Liquidate(_future.Mapped);
        }
    }
}
```

The follow table describes the arguments that the `G` method accepts in addition to the standard parameters :

Argument	Data Type	Description	Default Value
`ivModel`	`OptionPricingModelType`	The Option pricing model to use to estimate the IV when calculating Gamma. If you don't provide a value, the default value is to match the `optionModel` parameter.	`null`

For more information about the `G` method, see Using G Indicator .

Manual Indicators

To create a manual indicator for gamma, call the `Gamma` constructor.

```
C#

public class ManualGammaIndicatorAlgorithm : QCAlgorithm
```

```
{
    private Future _future;
    private ConstantDividendYieldModel _dividendYieldProvider;
    // Define the Option pricing model.
    private readonly OptionPricingModelType _optionPricingModel = OptionPricingModelType.ForwardTree;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _future = AddFuture(Futures.Indices.SP500EMini,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            dataMappingMode: DataMappingMode.OpenInterest,
            contractDepthOffset: 0);
        _future.SetFilter(0, 182);

        // Set up dividend yield provider for the underlying
        _dividendYieldProvider = new ConstantDividendYieldModel(0m);

        // Set up a Scheduled Event to select contract and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_future.Symbol),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_future.Mapped == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_future.Mapped);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddFutureOptionContract(contracts[0]);
                var contract2 = AddFutureOptionContract(contracts[1]);

                // Create and save the manual Gamma indicators.
                foreach (var (contractA, contractB) in new[] { (contract1, contract2), (contract2, contract1) })
                {
                    var option = contractA as dynamic;
                    option.Gamma = new Gamma(
                            contractA.Symbol, RiskFreeInterestRateModel, _dividendYieldProvider,
                            contractB.Symbol, _optionPricingModel
                        );
                }

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
```

```
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var option in chain)
            {
                var indicator = (Securities[option] as dynamic).Gamma;
                var underlying = option.Symbol.Underlying;
                var mirror = QuantConnect.Symbol.CreateOption(
                    underlying.Value, option.ID.Market, option.ID.OptionStyle,
                    option.ID.OptionRight == OptionRight.Call ? OptionRight.Put : OptionRight.Call,
                    option.ID.StrikePrice, option.ID.Date
                ).Value;

                // Check if price data is available for both contracts and the underlying asset.
                var q = slice.QuoteBars;
                var b = slice.Bars;
                if (q.ContainsKey(option) && q.ContainsKey(mirror) && b.ContainsKey(underlying))
                {
                    var dataPoints = new List<IndicatorDataPoint>
                    {
                        new IndicatorDataPoint(option, q[option].EndTime, q[option].Close),
                        new IndicatorDataPoint(mirror, q[mirror].EndTime, q[mirror].Close),
                        new IndicatorDataPoint(
                            underlying, b[underlying].EndTime, b[underlying].Close
                        )
                    };
                    foreach (var dataPoint in dataPoints)
                    {
                        indicator.Update(dataPoint);
                    }

                    // Get the current value of the Gamma indicator.
                    var value = indicator.Current.Value;
                }
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_future.Mapped].Invested)
        {
            Liquidate(_future.Mapped);
        }
    }
}
```

For more information about the `Gamma` constructor, see Using G Indicator .

Volatility Smoothing

The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, pass a `mirrorOption` argument to the indicator method or constructor and then call the `SetSmoothingFunction` method of the `ImpliedVolatility` property of the indicator.

```
// Example: Average IV of the call-put pair.
gammaIndicator.ImpliedVolatility.SetSmoothingFunction((iv, mirrorIv) => (iv + mirrorIv) * 0.5m);
```

Vega

Vega, ν , is the rate of change in the value of the Option with respect to the volatility of the underlying asset. For more information about vega, see Vega .

Automatic Indicators

To create an automatic indicator for vega, call the `QCAlgorithm.V` method with the Option contract `Symbol` object(s).

C#

```
public class AutomaticVegaIndicatorAlgorithm : QCAlgorithm
{
    private Future _future;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _future = AddFuture(Futures.Indices.SP500EMini,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            dataMappingMode: DataMappingMode.OpenInterest,
            contractDepthOffset: 0);
        _future.SetFilter(0, 182);

        // Set up a Scheduled Event to select contracts and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_future.Symbol),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_future.Mapped == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_future.Mapped);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count() > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddFutureOptionContract(contracts[0]) as dynamic;
                var contract2 = AddFutureOptionContract(contracts[1]) as dynamic;

                // Create and save the automatic Vega indicators.
                contract1.V = V(contracts[0], contracts[1]);
                contract2.V = V(contracts[1], contracts[0]);

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        // Get the Vega indicator of each contract.
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var symbol in chain)
            {
                var option = Securities[symbol] as dynamic;
                var indicator = option.V;
```

```
                var indicator = option.V;
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_future.Mapped].Invested)
        {
            Liquidate(_future.Mapped);
        }
    }
}
```

The follow table describes the arguments that the `V` method accepts in addition to the standard parameters :

Argument	Data Type	Description	Default Value
`ivModel`	`OptionPricingModelType`	The Option pricing model to use to estimate the IV when calculating Vega. If you don't provide a value, the default value is to match the `optionModel` parameter.	`null`

For more information about the `V` method, see Using V Indicator .

Manual Indicators

To create a manual indicator for vega, call the `Vega` constructor.

C#

```
public class ManualVegaIndicatorAlgorithm : QCAlgorithm
{
    private Future _future;
    private ConstantDividendYieldModel _dividendYieldProvider;
    // Define the Option pricing model.
    private readonly OptionPricingModelType _optionPricingModel = OptionPricingModelType.ForwardTree;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _future = AddFuture(Futures.Indices.SP500EMini,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            dataMappingMode: DataMappingMode.OpenInterest,
            contractDepthOffset: 0);
        _future.SetFilter(0, 182);

        // Set up dividend yield provider for the underlying
        _dividendYieldProvider = new ConstantDividendYieldModel(0m);

        // Set up a Scheduled Event to select contract and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_future.Symbol),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_future.Mapped == null)
```

```
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_future.Mapped);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddFutureOptionContract(contracts[0]);
                var contract2 = AddFutureOptionContract(contracts[1]);

                // Create and save the manual Vega indicators.
                foreach (var (contractA, contractB) in new[] { (contract1, contract2), (contract2, contract1) })
                {
                    var option = contractA as dynamic;
                    option.Vega = new Vega(
                            contractA.Symbol, RiskFreeInterestRateModel, _dividendYieldProvider,
                            contractB.Symbol, _optionPricingModel
                        );
                }

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var option in chain)
            {
                var indicator = (Securities[option] as dynamic).Vega;
                var underlying = option.Symbol.Underlying;
                var mirror = QuantConnect.Symbol.CreateOption(
                    underlying.Value, option.ID.Market, option.ID.OptionStyle,
                    option.ID.OptionRight == OptionRight.Call ? OptionRight.Put : OptionRight.Call,
                    option.ID.StrikePrice, option.ID.Date
                ).Value;

                // Check if price data is available for both contracts and the underlying asset.
                var q = slice.QuoteBars;
                var b = slice.Bars;
                if (q.ContainsKey(option) && q.ContainsKey(mirror) && b.ContainsKey(underlying))
                {
                    var dataPoints = new List<IndicatorDataPoint>
                    {
                        new IndicatorDataPoint(option, q[option].EndTime, q[option].Close),
                        new IndicatorDataPoint(mirror, q[mirror].EndTime, q[mirror].Close),
                        new IndicatorDataPoint(
                            underlying, b[underlying].EndTime, b[underlying].Close
                        )
                    };
                    foreach (var dataPoint in dataPoints)
                    {
                        indicator.Update(dataPoint);
                    }

                    // Get the current value of the Vega indicator.
                    var value = indicator.Current.Value;
                }
            }
        }
```

```
        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_future.Mapped].Invested)
        {
            Liquidate(_future.Mapped);
        }
    }
}
```

For more information about the `Vega` constructor, see Using V Indicator .

Volatility Smoothing

The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, pass a `mirrorOption` argument to the indicator method or constructor and then call the `SetSmoothingFunction` method of the `ImpliedVolatility` property of the indicator.

C#

```
// Example: Average IV of the call-put pair.
vegaIndicator.ImpliedVolatility.SetSmoothingFunction((iv, mirrorIv) => (iv + mirrorIv) * 0.5m);
```

Theta

Theta, Θ , is the rate of change of the value of the Option with respect to the passage of time. It is also known to as the time decay of an Option. For more information about theta, see Theta .

Automatic Indicators

To create an automatic indicator for theta, call the `QCAlgorithm.T` method with the Option contract `Symbol` object(s).

C#

```
public class AutomaticThetaIndicatorAlgorithm : QCAlgorithm
{
    private Future _future;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _future = AddFuture(Futures.Indices.SP500EMini,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            dataMappingMode: DataMappingMode.OpenInterest,
            contractDepthOffset: 0);
        _future.SetFilter(0, 182);

        // Set up a Scheduled Event to select contracts and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_future.Symbol),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_future.Mapped == null)
        {
            return;
```

```
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_future.Mapped);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count() > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddFutureOptionContract(contracts[0]) as dynamic;
                var contract2 = AddFutureOptionContract(contracts[1]) as dynamic;

                // Create and save the automatic Theta indicators.
                contract1.T = T(contracts[0], contracts[1]);
                contract2.T = T(contracts[1], contracts[0]);

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        // Get the Theta indicator of each contract.
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var symbol in chain)
            {
                var option = Securities[symbol] as dynamic;
                var indicator = option.T;
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_future.Mapped].Invested)
        {
            Liquidate(_future.Mapped);
        }
    }
}
```

The follow table describes the arguments that the `T` method accepts in addition to the standard parameters :

Argument	Data Type	Description	Default Value
`ivModel`	`OptionPricingModelType`	The Option pricing model to use to estimate the IV when calculating theta. If you don't provide a value, the default value is to match the `optionModel` parameter.	`null`

For more information about the T method, see Using T Indicator .

Manual Indicators

To create a manual indicator for theta, call the Theta constructor.

C#

```
public class ManualThetaIndicatorAlgorithm : QCAlgorithm
{
    private Future _future;
    private ConstantDividendYieldModel _dividendYieldProvider;
    // Define the Option pricing model.
    private readonly OptionPricingModelType _optionPricingModel = OptionPricingModelType.ForwardTree;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _future = AddFuture(Futures.Indices.SP500EMini,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            dataMappingMode: DataMappingMode.OpenInterest,
            contractDepthOffset: 0);
        _future.SetFilter(0, 182);

        // Set up dividend yield provider for the underlying
        _dividendYieldProvider = new ConstantDividendYieldModel(0m);

        // Set up a Scheduled Event to select contract and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_future.Symbol),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_future.Mapped == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_future.Mapped);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddFutureOptionContract(contracts[0]);
                var contract2 = AddFutureOptionContract(contracts[1]);

                // Create and save the manual Theta indicators.
                foreach (var (contractA, contractB) in new[] { (contract1, contract2), (contract2, contract1) })
                {
                    var option = contractA as dynamic;
                    option.Theta = new Theta(
                            contractA.Symbol, RiskFreeInterestRateModel, _dividendYieldProvider,
```

```
                            contractB.Symbol, _optionPricingModel
                        );
                }

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var option in chain)
            {
                var indicator = (Securities[option] as dynamic).Theta;
                var underlying = option.Symbol.Underlying;
                var mirror = QuantConnect.Symbol.CreateOption(
                    underlying.Value, option.ID.Market, option.ID.OptionStyle,
                    option.ID.OptionRight == OptionRight.Call ? OptionRight.Put : OptionRight.Call,
                    option.ID.StrikePrice, option.ID.Date
                ).Value;

                // Check if price data is available for both contracts and the underlying asset.
                var q = slice.QuoteBars;
                var b = slice.Bars;
                if (q.ContainsKey(option) && q.ContainsKey(mirror) && b.ContainsKey(underlying))
                {
                    var dataPoints = new List<IndicatorDataPoint>
                    {
                        new IndicatorDataPoint(option, q[option].EndTime, q[option].Close),
                        new IndicatorDataPoint(mirror, q[mirror].EndTime, q[mirror].Close),
                        new IndicatorDataPoint(
                            underlying, b[underlying].EndTime, b[underlying].Close
                        )
                    };
                    foreach (var dataPoint in dataPoints)
                    {
                        indicator.Update(dataPoint);
                    }

                    // Get the current value of the Theta indicator.
                    var value = indicator.Current.Value;
                }
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_future.Mapped].Invested)
        {
            Liquidate(_future.Mapped);
        }
    }
}
```

For more information about the `Theta` constructor, see Using T Indicator .

Volatility Smoothing

The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, pass a `mirrorOption` argument to the indicator method or constructor and then call the `SetSmoothingFunction` method of the `ImpliedVolatility` property of the indicator.

```
C#
// Example: Average IV of the call-put pair.
thetaIndicator.ImpliedVolatility.SetSmoothingFunction((iv, mirrorIv) => (iv + mirrorIv) * 0.5m);
```

Rho

Rho, ρ, is the rate of change of the value of a derivative with respect to the interest rate. It is usually small and not a big issue in practice unless the Option is deep in-the-money and has a long horizon. In this case, the interest rate matters because you need to discount a larger cash flow over a longer horizon. For more information about rho, see Rho.

Automatic Indicators

To create an automatic indicator for rho, call the `QCAlgorithm.R` method with the Option contract `Symbol` object(s).

```
C#
public class AutomaticRhoIndicatorAlgorithm : QCAlgorithm
{
    private Future _future;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _future = AddFuture(Futures.Indices.SP500EMini,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            dataMappingMode: DataMappingMode.OpenInterest,
            contractDepthOffset: 0);
        _future.SetFilter(0, 182);

        // Set up a Scheduled Event to select contracts and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_future.Symbol),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_future.Mapped == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_future.Mapped);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count() > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddFutureOptionContract(contracts[0]) as dynamic;
                var contract2 = AddFutureOptionContract(contracts[1]) as dynamic;

                // Create and save the automatic Rho indicators.
                contract1.R = R(contracts[0], contracts[1]);
                contract2.R = R(contracts[1], contracts[0]);

                _options = (contracts[0], contracts[1]);
            }
        }
```

```
        }
    }

    public override void OnData(Slice slice)
    {
        // Get the Rho indicator of each contract.
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var symbol in chain)
            {
                var option = Securities[symbol] as dynamic;
                var indicator = option.R;
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_future.Mapped].Invested)
        {
            Liquidate(_future.Mapped);
        }
    }
}
```

The follow table describes the arguments that the `R` method accepts in addition to the standard parameters :

Argument	Data Type	Description	Default Value
`ivModel`	`OptionPricingModelType`	The Option pricing model to use to estimate the IV when calculating rho If you don't provide a value, the default value is to match the `optionModel` parameter.	`null`

For more information about the `R` method, see Using R Indicator .

Manual Indicators

To create a manual indicator for rho, call the `Rho` constructor.

C#

```
public class ManualRhoIndicatorAlgorithm : QCAlgorithm
{
    private Future _future;
    private ConstantDividendYieldModel _dividendYieldProvider;
    // Define the Option pricing model.
    private readonly OptionPricingModelType _optionPricingModel = OptionPricingModelType.ForwardTree;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _future = AddFuture(Futures.Indices.SP500EMini,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            dataMappingMode: DataMappingMode.OpenInterest,
            contractDepthOffset: 0);
        _future.SetFilter(0, 182);

        // Set up dividend yield provider for the underlying
        _dividendYieldProvider = new ConstantDividendYieldModel(0m);

        // Set up a Scheduled Event to select contract and create the indicators every day before market open.
```

```
        Schedule.On(
            DateRules.EveryDay(_future.Symbol),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_future.Mapped == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_future.Mapped);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddFutureOptionContract(contracts[0]);
                var contract2 = AddFutureOptionContract(contracts[1]);

                // Create and save the manual Rho indicators.
                foreach (var (contractA, contractB) in new[] { (contract1, contract2), (contract2, contract1) })
                {
                    var option = contractA as dynamic;
                    option.Rho = new Rho(
                            contractA.Symbol, RiskFreeInterestRateModel, _dividendYieldProvider,
                            contractB.Symbol, _optionPricingModel
                        );
                }

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var option in chain)
            {
                var indicator = (Securities[option] as dynamic).Rho;
                var underlying = option.Symbol.Underlying;
                var mirror = QuantConnect.Symbol.CreateOption(
                    underlying.Value, option.ID.Market, option.ID.OptionStyle,
                    option.ID.OptionRight == OptionRight.Call ? OptionRight.Put : OptionRight.Call,
                    option.ID.StrikePrice, option.ID.Date
                ).Value;

                // Check if price data is available for both contracts and the underlying asset.
                var q = slice.QuoteBars;
                var b = slice.Bars;
                if (q.ContainsKey(option) && q.ContainsKey(mirror) && b.ContainsKey(underlying))
                {
                    var dataPoints = new List<IndicatorDataPoint>
                    {
                        new IndicatorDataPoint(option, q[option].EndTime, q[option].Close),
                        new IndicatorDataPoint(mirror, q[mirror].EndTime, q[mirror].Close),
                        new IndicatorDataPoint(
                            underlying, b[underlying].EndTime, b[underlying].Close
                        )
                    };
```

```
                    foreach (var dataPoint in dataPoints)
                    {
                        indicator.Update(dataPoint);
                    }

                    // Get the current value of the Rho indicator.
                    var value = indicator.Current.Value;
                }
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_future.Mapped].Invested)
        {
            Liquidate(_future.Mapped);
        }
    }
}
```

For more information about the `Rho` constructor, see Using R Indicator .

Volatility Smoothing

The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, pass a `mirrorOption` argument to the indicator method or constructor and then call the `SetSmoothingFunction` method of the `ImpliedVolatility` property of the indicator.

C#

```
// Example: Average IV of the call-put pair.
rhoIndicator.ImpliedVolatility.SetSmoothingFunction((iv, mirrorIv) => (iv + mirrorIv) * 0.5m);
```

Examples

Demonstration Algorithms
OptionIndicatorsRegressionAlgorithm.cs C# OptionIndicatorsMirrorContractsRegressionAlgorithm.cs C#

Future Options

Market Hours

Introduction

The Future Option markets have the same hours as the Futures market. For more information, see the supported contracts and the market hours of Future markets .

Asset Classes

Index

There are cash Indices and volatility Indices. The former is a hypothetical portfolio that represents a segment of the financial market and the latter shows the expected level of price fluctuation in an Index Option.

Requesting Data

Handling Data

Market Hours

See Also

BasicTemplateIndexAlgorithm.py
BasicTemplateIndexAlgorithm.cs

Index

Requesting Data

Introduction

Request Index data in your algorithm to receive a feed of Index prices in the `OnData` method. For more information about the specific dataset we use for backtests, see the US Cash Indices dataset listing . To trade live with Index data, you can use one of the brokerage data providers .

Create Subscriptions

To create an Index subscription, in the `Initialize` method, call the `AddIndex` method. The `AddIndex` method returns an `Index` object, which contains a `Symbol` property. Save a reference to the `Symbol` so you can use it in `OnData` to access the security data in the `Slice` .

```
C#
_symbol = AddIndex("VIX").Symbol;
```

To view the supported assets in the US Cash Indices dataset, see the Supported Indices .

Resolutions

The following table shows the available resolutions and data formats for Index subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`				
`Second`				
`Minute`	✓			
`Hour`	✓			
`Daily`	✓			

The default resolution for Index subscriptions is `Resolution.Minute` . To change the resolution, pass a `resolution` argument to the `AddIndex` method.

```
C#
_symbol = AddIndex("VIX", Resolution.Daily).Symbol;
```

To create custom resolution periods, see Consolidating Data .

Supported Markets

The only market available for Indices is `Market.USA` , so you don't need to pass a `market` argument to the `AddIndex` method.

```
_symbol = AddIndex("VIX", market: Market.USA).Symbol;
```

The brokerage models have a default market for each asset class. If you set a brokerage model, you may not need to specify the market to use.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
_symbol = AddIndex("VIX", fillForward: false).Symbol;
```

Data Normalization

The data normalization mode doesn't affect the data that LEAN passes to `OnData` or the data from history request . If you change the data normalization mode, it won't change the outcome.

Properties

The `AddIndex` method returns an `Index` object, which have the following properties:

Examples

The following examples demonstrate some common practices for requesting Index data.

Example 1: VIX EMA Cross

The following algorithm adds daily VIX Index data and uses the Exponential Moving Average (EMA) indicator to create trading signals. The algorithm uses a 30-day slow EMA and a 15-day fast EMA. It enters a long SPY position when the fast EMA crosses above the slow EMA and liquidates the position when the fast EMA crosses below the slow EMA.

C#

```
public class IndexDemoAlgorithm : QCAlgorithm
{
    private ExponentialMovingAverage _emaFast, _emaSlow;
    private Symbol _vix, _spy;
    private decimal _tolerance;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add the daily VIX and SPY data.
        _vix = AddIndex("VIX", Resolution.Daily).Symbol;
        _spy = AddEquity("SPY", Resolution.Daily).Symbol;
        // Create fast and slow EMA indicators.
        _emaFast = EMA(_vix, 15);
        _emaSlow = EMA(_vix, 30);
        // Define a small tolerance to reduce trading churn.
        _tolerance = 0.00015m;
    }

    public override void OnData(Slice data)
    {
        // Check if the indicators is ready.
        if (!(_emaSlow.IsReady))
        {
            return;
        }
        // When the fast VIX EMA crosses above the slow EMA, buy SPY.
        if (!Portfolio.Invested && (_emaFast.Current.Value > _emaSlow.Current.Value  * (1 + _tolerance)))
        {
            SetHoldings(_spy, 0.8m);
        }
        // When the fast VIX EMA crosses below the slow EMA, liquidate the position.
        else if (Portfolio.Invested && (_emaFast.Current.Value < _emaSlow.Current.Value  * (1 - _tolerance)))
        {
            Liquidate(_spy);
        }
    }
}
```

Index

Handling Data

Introduction

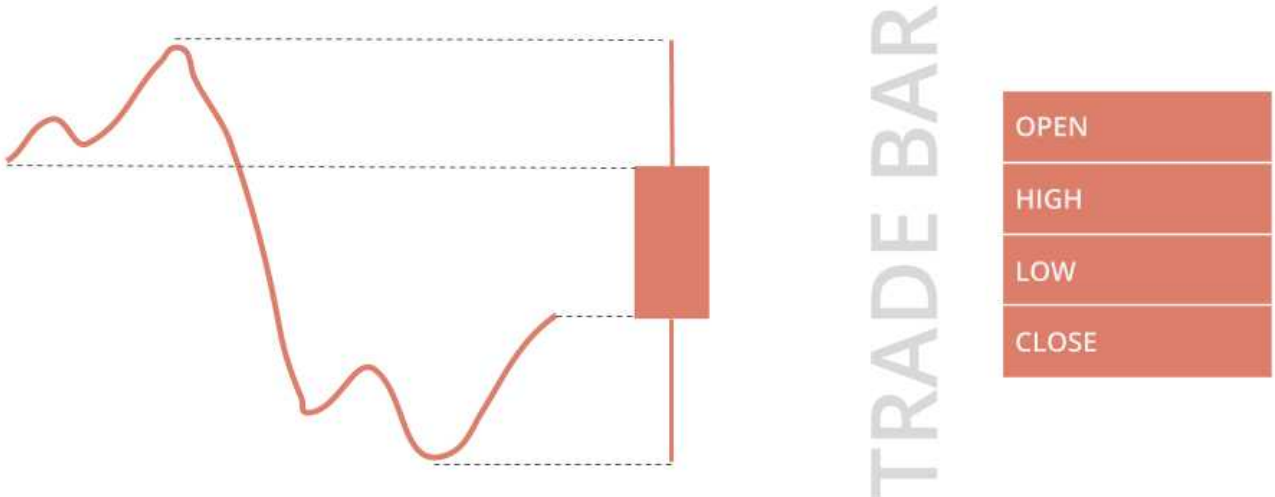
LEAN passes the data you request to the `OnData` method so you can make trading decisions. The default `OnData` method accepts a `Slice` object, but you can define additional `OnData` methods that accept different data types. For example, if you define an `OnData` method that accepts a `Tick` argument, it only receives `Tick` objects. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `Ticks DataDictionary` is made up of `Tick` objects. To access individual data points in the dictionary, you can index the dictionary with the Index ticker or `Symbol` , but we recommend you use the `Symbol` .

To view the resolutions that are available for Index data, see Resolutions .

Bars

You can't trade Indices, but `TradeBar` objects are bars that represent the open, high, low, and close of an Index price over a period of time.



To get the `TradeBar` objects in the `Slice` , index the `Slice` or index the `Bars` property of the `Slice` with the Index `Symbol` . The `Slice` may not contain data for your `Symbol` at every time step. To avoid issues, check if the `Slice` contains data for your Index before you index the `Slice` with the Index `Symbol` .

```C#
public override void OnData(Slice slice)
{
    if (slice.Bars.ContainsKey(_symbol))
    {
        var tradeBar = slice.Bars[_symbol];
        var value = tradeBar.Value;
    }
}
```

`TradeBar` objects have the following properties:

Examples

The following examples demonstrate some common practices for handling Index data.

Example 1: Trading ES Futures Based on the SPX Index RSI

The following algorithm adds daily data for the SPX Index and uses the Relative Strength Index (RSI) indicator to generate trading signals for the E-Mini S&P 500 Future. When the RSI crosses below 30, it buys the front-month ES contract. When the RSI crosses above 70, it liquidates the portfolio.

```C#
public class IndexDemoAlgorithm : QCAlgorithm
{
    private dynamic _spx;
    private Future _es;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add daily SPX Index data.
        _spx = AddIndex("SPX", Resolution.Daily);
        // Create an RSI indicator for the SPX index.
        _spx.Rsi = RSI(_spx.Symbol, 15);
        // Add the ES front-month Futures contract.
        _es = AddFuture(Futures.Indices.SP500EMini, dataMappingMode: DataMappingMode.OpenInterest);
        _es.SetFilter(futureFilterUniverse => futureFilterUniverse.StandardsOnly().FrontMonth());
    }

    public override void OnData(Slice slice)
    {
        // Check if the indicator is ready and we have the front-month contract data.
        if (!_spx.Rsi.IsReady || !slice.Bars.ContainsKey(_es.Mapped))
        {
            return;
        }

        // If the SPX RSI is <30, buy the ES.
        if (!Portfolio.Invested && _spx.Rsi.Current.Value < 30)
        {
            SetHoldings(_es.Mapped, 0.5);
        }

        // If the SPX RSI is >70, liquidate the portfolio.
        else if (Portfolio.Invested && _spx.Rsi.Current.Value > 70)
        {
            Liquidate();
        }
    }
}
```

Index

Market Hours

USA

EUREX

HKFE

OSE

Market Hours

USA

Introduction

This page shows the trading hours, holidays, and time zone of the US Indices market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US Indices market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:20:00
Tuesday	08:30:00 to 15:20:00
Wednesday	08:30:00 to 15:20:00
Thursday	08:30:00 to 15:20:00
Friday	08:30:00 to 15:20:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

The following table shows the early closes for the US Indices market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/Chicago)**
1999-11-26	12:00:00
2000-07-03	12:00:00
2000-11-24	12:00:00
2001-07-03	12:00:00
2001-11-23	12:00:00
2001-12-24	12:00:00

2002-07-05	12:00:00
2002-11-29	12:00:00
2002-12-24	12:00:00
2003-07-03	12:00:00
2003-11-28	12:00:00
2003-12-24	12:00:00
2003-12-26	12:00:00
2004-11-26	12:00:00
2005-11-25	12:00:00
2006-07-03	12:00:00
2006-11-24	12:00:00
2007-07-03	12:00:00
2007-11-23	12:00:00
2007-12-24	12:00:00
2008-07-03	12:00:00
2008-11-28	12:00:00
2008-12-24	12:00:00
2009-11-27	12:00:00
2009-12-24	12:00:00
2010-11-26	12:00:00
2011-11-25	12:00:00
2012-07-03	12:00:00
2012-11-23	12:00:00
2012-12-24	12:00:00
2013-07-03	12:00:00
2013-11-29	12:00:00
2013-12-24	12:00:00
2014-07-03	12:00:00

2014-11-28	12:00:00
2014-12-24	12:00:00
2015-11-27	12:00:00
2015-12-24	12:00:00
2016-11-25	12:00:00
2017-07-03	12:00:00
2017-11-24	12:00:00
2017-12-24	12:00:00
2018-07-03	12:00:00
2018-11-23	12:00:00
2018-12-24	12:00:00
2019-07-03	12:00:00
2019-11-29	12:00:00
2019-12-24	12:00:00
2020-11-27	12:00:00
2020-12-24	12:00:00
2021-11-26	12:00:00
2022-11-25	12:00:00
2023-07-03	12:00:00
2023-11-24	12:00:00
2024-07-03	12:00:00
2024-11-29	12:00:00
2024-12-24	12:00:00
2025-07-03	12:00:00
2025-11-28	12:00:00
2025-12-24	12:00:00
2026-11-27	12:00:00
2026-12-24	12:00:00

Late Opens

There are no days with late opens.

Time Zone

The US Indices market trades in the following time zones:

- `America/Chicago`
- `America/Los Angeles`
- `America/New York`
- `Europe/Berlin`

Assets With Other Hours

The following table shows the indices that have different trading periods than the overall US Indices market:

Symbol	Name
AEX	AEX
BKX	BKX
BRR	BRR
BRTI	BRTI
BXD	BXD
BXM	BXM
BXN	BXN
BXR	BXR
CEX	CEX
CLL	CLL
COMP	COMP
COR1M	COR1M
COR1Y	COR1Y
COR30D	COR30D
COR3M	COR3M
COR6M	COR6M
COR9M	COR9M
DJCIAGC	DJCIAGC
DJCICC	DJCICC

DJCIGC	DJCIGC
DJCIGR	DJCIGR
DJCIIK	DJCIIK
DJCIKC	DJCIKC
DJCISB	DJCISB
DJCISI	DJCISI
DJI	DJI
DJINET	DJINET
DJR	DJR
DJX	DJX
DRG	DRG
DTX	DTX
DUX	DUX
DVS	DVS
DWCF	DWCF
DWCPF	DWCPF
DXL	DXL
DXY	DXY
EVZ	EVZ
FTW5000	FTW5000
FVX	FVX
GVZ	GVZ
HGX	HGX
MID	MID
MIDG	MIDG
MIDV	MIDV
MRUT	MRUT
NDX	Nasdaq 100 Index

NYA	NYA
NYFANG	NYFANG
NYXBT	NYXBT
OEX	OEX
OSX	OSX
OVX	OVX
PUT	PUT
RLS	RLS
RUA	RUA
RUI	RUI
RUT	Russell 2000 Index
RVX	RVX
SET	SET
SGX	SGX
SKEW	SKEW
SMLG	SMLG
SP600	SP600
SPGSCI	SPGSCI
SPSIBI	SPSIBI
SPSV	SPSV
SPX	S&P 500 Index
SVX	SVX
TNX	TNX
TYX	TYX
UKX	UKX
UTIL	UTIL
UTY	UTY
VAF	VAF

VIF	VIF
VIN	VIN
VIX	CBOE Volatility Index
VIX1D	VIX1D
VIX1Y	VIX1Y
VIX3M	VIX3M
VIX6M	VIX6M
VIX9D	VIX9D
VOLI	VOLI
VPD	VPD
VPN	VPN
VRO	VRO
VVIX	VVIX
VWA	VWA
VWB	VWB
VXD	VXD
VXN	VXN
VXO	VXO
VXSLV	VXSLV
VXTH	VXTH
VXTLT	VXTLT
XAU	XAU
XDA	XDA
XDB	XDB
XEO	XEO
XMI	XMI
XNDX	XNDX
XSP	XSP

USA

AEX

Introduction

This page shows the trading hours, holidays, and time zone of the AEX market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the AEX market:

Weekday	**Time (Europe/Berlin)**
Monday	09:00:00 to 17:40:00
Tuesday	09:00:00 to 17:40:00
Wednesday	09:00:00 to 17:40:00
Thursday	09:00:00 to 17:40:00
Friday	09:00:00 to 17:40:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The AEX market trades in the `Europe/Berlin` time zone.

USA

BKX

Introduction

This page shows the trading hours, holidays, and time zone of the BKX market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the BKX market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:00:00
Tuesday	09:30:00 to 16:00:00
Wednesday	09:30:00 to 16:00:00
Thursday	09:30:00 to 16:00:00
Friday	09:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The BKX market trades in the `America/New York` time zone.

USA

BRR

Introduction

This page shows the trading hours, holidays, and time zone of the BRR market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the BRR market:

Weekday	**Time (America/Chicago)**
Sunday	00:00:00 to 23:58:00
Monday	00:00:00 to 23:58:00
Tuesday	00:00:00 to 23:58:00
Wednesday	00:00:00 to 23:58:00
Thursday	00:00:00 to 23:58:00
Friday	00:00:00 to 23:58:00
Saturday	00:00:00 to 23:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13

2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18

2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The BRR market trades in the `America/Chicago` time zone.

USA

BRTI

Introduction

This page shows the trading hours, holidays, and time zone of the BRTI market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the BRTI market:

Weekday	**Time (America/Chicago)**
Sunday	00:00:00 to 23:58:00
Monday	00:00:00 to 23:58:00
Tuesday	00:00:00 to 23:58:00
Wednesday	00:00:00 to 23:58:00
Thursday	00:00:00 to 23:58:00
Friday	00:00:00 to 23:58:00
Saturday	00:00:00 to 23:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13

2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18

2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The BRTI market trades in the `America/Chicago` time zone.

USA

BXD

Introduction

This page shows the trading hours, holidays, and time zone of the BXD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the BXD market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The BXD market trades in the `America/Chicago` time zone.

USA

BXM

Introduction

This page shows the trading hours, holidays, and time zone of the BXM market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the BXM market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The BXM market trades in the `America/Chicago` time zone.

USA

BXN

Introduction

This page shows the trading hours, holidays, and time zone of the BXN market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the BXN market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The BXN market trades in the `America/Chicago` time zone.

USA

BXR

Introduction

This page shows the trading hours, holidays, and time zone of the BXR market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the BXR market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The BXR market trades in the `America/Chicago` time zone.

USA

CEX

Introduction

This page shows the trading hours, holidays, and time zone of the CEX market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CEX market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CEX market trades in the `America/Chicago` time zone.

USA

CLL

Introduction

This page shows the trading hours, holidays, and time zone of the CLL market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CLL market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CLL market trades in the `America/Chicago` time zone.

USA

COMP

Introduction

This page shows the trading hours, holidays, and time zone of the COMP market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the COMP market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:00:00
Tuesday	09:30:00 to 16:00:00
Wednesday	09:30:00 to 16:00:00
Thursday	09:30:00 to 16:00:00
Friday	09:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The COMP market trades in the `America/New York` time zone.

USA

COR1M

Introduction

This page shows the trading hours, holidays, and time zone of the COR1M market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the COR1M market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The COR1M market trades in the `America/Chicago` time zone.

USA

COR1Y

Introduction

This page shows the trading hours, holidays, and time zone of the COR1Y market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the COR1Y market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The COR1Y market trades in the `America/Chicago` time zone.

USA

COR30D

Introduction

This page shows the trading hours, holidays, and time zone of the COR30D market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the COR30D market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The COR30D market trades in the `America/Chicago` time zone.

USA

COR3M

Introduction

This page shows the trading hours, holidays, and time zone of the COR3M market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the COR3M market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The COR3M market trades in the `America/Chicago` time zone.

USA

COR6M

Introduction

This page shows the trading hours, holidays, and time zone of the COR6M market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the COR6M market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The COR6M market trades in the `America/Chicago` time zone.

USA

COR9M

Introduction

This page shows the trading hours, holidays, and time zone of the COR9M market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the COR9M market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The COR9M market trades in the `America/Chicago` time zone.

USA

DJCIAGC

Introduction

This page shows the trading hours, holidays, and time zone of the DJCIAGC market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DJCIAGC market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DJCIAGC market trades in the `America/Chicago` time zone.

USA

DJCICC

Introduction

This page shows the trading hours, holidays, and time zone of the DJCICC market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DJCICC market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DJCICC market trades in the `America/Chicago` time zone.

USA

DJCIGC

Introduction

This page shows the trading hours, holidays, and time zone of the DJCIGC market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DJCIGC market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DJCIGC market trades in the `America/Chicago` time zone.

USA

DJCIGR

Introduction

This page shows the trading hours, holidays, and time zone of the DJCIGR market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DJCIGR market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DJCIGR market trades in the `America/Chicago` time zone.

USA

DJCIIK

Introduction

This page shows the trading hours, holidays, and time zone of the DJCIIK market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DJCIIK market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DJCIIK market trades in the `America/Chicago` time zone.

USA

DJCIKC

Introduction

This page shows the trading hours, holidays, and time zone of the DJCIKC market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DJCIKC market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DJCIKC market trades in the `America/Chicago` time zone.

USA

DJCISB

Introduction

This page shows the trading hours, holidays, and time zone of the DJCISB market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DJCISB market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DJCISB market trades in the `America/Chicago` time zone.

USA

DJCISI

Introduction

This page shows the trading hours, holidays, and time zone of the DJCISI market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DJCISI market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DJCISI market trades in the `America/Chicago` time zone.

USA

DJI

Introduction

This page shows the trading hours, holidays, and time zone of the DJI market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DJI market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:15:00
Tuesday	08:30:00 to 15:15:00
Wednesday	08:30:00 to 15:15:00
Thursday	08:30:00 to 15:15:00
Friday	08:30:00 to 15:15:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DJI market trades in the `America/Chicago` time zone.

USA

DJINET

Introduction

This page shows the trading hours, holidays, and time zone of the DJINET market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DJINET market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DJINET market trades in the `America/Chicago` time zone.

USA

DJR

Introduction

This page shows the trading hours, holidays, and time zone of the DJR market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DJR market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DJR market trades in the `America/Chicago` time zone.

USA

DJX

Introduction

This page shows the trading hours, holidays, and time zone of the DJX market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DJX market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DJX market trades in the `America/Chicago` time zone.

USA

DRG

Introduction

This page shows the trading hours, holidays, and time zone of the DRG market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DRG market:

Weekday	**Time (America/Los Angeles)**
Monday	06:30:00 to 13:00:00
Tuesday	06:30:00 to 13:00:00
Wednesday	06:30:00 to 13:00:00
Thursday	06:30:00 to 13:00:00
Friday	06:30:00 to 13:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DRG market trades in the `America/Los Angeles` time zone.

USA

DTX

Introduction

This page shows the trading hours, holidays, and time zone of the DTX market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DTX market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DTX market trades in the `America/Chicago` time zone.

USA

DUX

Introduction

This page shows the trading hours, holidays, and time zone of the DUX market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DUX market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DUX market trades in the `America/Chicago` time zone.

USA

DVS

Introduction

This page shows the trading hours, holidays, and time zone of the DVS market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DVS market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DVS market trades in the `America/Chicago` time zone.

USA

DWCF

Introduction

This page shows the trading hours, holidays, and time zone of the DWCF market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DWCF market:

Weekday	**Time (America/Los Angeles)**
Monday	06:30:00 to 13:00:00
Tuesday	06:30:00 to 13:00:00
Wednesday	06:30:00 to 13:00:00
Thursday	06:30:00 to 13:00:00
Friday	06:30:00 to 13:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DWCF market trades in the `America/Los Angeles` time zone.

USA

DWCPF

Introduction

This page shows the trading hours, holidays, and time zone of the DWCPF market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DWCPF market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DWCPF market trades in the `America/Chicago` time zone.

USA

DXL

Introduction

This page shows the trading hours, holidays, and time zone of the DXL market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DXL market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DXL market trades in the `America/Chicago` time zone.

USA

DXY

Introduction

This page shows the trading hours, holidays, and time zone of the DXY market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DXY market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Friday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DXY market trades in the `America/New York` time zone.

USA

EVZ

Introduction

This page shows the trading hours, holidays, and time zone of the EVZ market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EVZ market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EVZ market trades in the `America/Chicago` time zone.

USA

FTW5000

Introduction

This page shows the trading hours, holidays, and time zone of the FTW5000 market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the FTW5000 market:

Weekday	**Time (America/New York)**
Monday	07:00:00 to 20:00:00
Tuesday	07:00:00 to 20:00:00
Wednesday	07:00:00 to 20:00:00
Thursday	07:00:00 to 20:00:00
Friday	07:00:00 to 20:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The FTW5000 market trades in the `America/New York` time zone.

USA

FVX

Introduction

This page shows the trading hours, holidays, and time zone of the FVX market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the FVX market:

Weekday	**Time (America/Chicago)**
Monday	07:20:00 to 14:00:00
Tuesday	07:20:00 to 14:00:00
Wednesday	07:20:00 to 14:00:00
Thursday	07:20:00 to 14:00:00
Friday	07:20:00 to 14:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The FVX market trades in the `America/Chicago` time zone.

USA

GVZ

Introduction

This page shows the trading hours, holidays, and time zone of the GVZ market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GVZ market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The GVZ market trades in the `America/Chicago` time zone.

USA

HGX

Introduction

This page shows the trading hours, holidays, and time zone of the HGX market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the HGX market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:00:00
Tuesday	09:30:00 to 16:00:00
Wednesday	09:30:00 to 16:00:00
Thursday	09:30:00 to 16:00:00
Friday	09:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The HGX market trades in the `America/New York` time zone.

USA

MID

Introduction

This page shows the trading hours, holidays, and time zone of the MID market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the MID market:

Weekday	**Time (America/Los Angeles)**
Monday	06:30:00 to 13:00:00
Tuesday	06:30:00 to 13:00:00
Wednesday	06:30:00 to 13:00:00
Thursday	06:30:00 to 13:00:00
Friday	06:30:00 to 13:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The MID market trades in the `America/Los Angeles` time zone.

USA

MIDG

Introduction

This page shows the trading hours, holidays, and time zone of the MIDG market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the MIDG market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The MIDG market trades in the `America/Chicago` time zone.

USA

MIDV

Introduction

This page shows the trading hours, holidays, and time zone of the MIDV market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the MIDV market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The MIDV market trades in the `America/Chicago` time zone.

USA

MRUT

Introduction

This page shows the trading hours, holidays, and time zone of the MRUT market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the MRUT market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The MRUT market trades in the `America/Chicago` time zone.

USA

NDX

Introduction

This page shows the trading hours, holidays, and time zone of the Nasdaq 100 Index market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Nasdaq 100 Index market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:00:00
Tuesday	09:30:00 to 16:00:00
Wednesday	09:30:00 to 16:00:00
Thursday	09:30:00 to 16:00:00
Friday	09:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Nasdaq 100 Index market trades in the `America/New York` time zone.

USA

NYA

Introduction

This page shows the trading hours, holidays, and time zone of the NYA market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NYA market:

Weekday	**Time (America/Los Angeles)**
Monday	06:30:00 to 13:00:00
Tuesday	06:30:00 to 13:00:00
Wednesday	06:30:00 to 13:00:00
Thursday	06:30:00 to 13:00:00
Friday	06:30:00 to 13:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The NYA market trades in the `America/Los Angeles` time zone.

USA

NYFANG

Introduction

This page shows the trading hours, holidays, and time zone of the NYFANG market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NYFANG market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:00:00
Tuesday	09:30:00 to 16:00:00
Wednesday	09:30:00 to 16:00:00
Thursday	09:30:00 to 16:00:00
Friday	09:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The NYFANG market trades in the `America/New York` time zone.

USA

NYXBT

Introduction

This page shows the trading hours, holidays, and time zone of the NYXBT market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NYXBT market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:00:00
Tuesday	09:30:00 to 16:00:00
Wednesday	09:30:00 to 16:00:00
Thursday	09:30:00 to 16:00:00
Friday	09:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The NYXBT market trades in the `America/New York` time zone.

USA

OEX

Introduction

This page shows the trading hours, holidays, and time zone of the OEX market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the OEX market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The OEX market trades in the `America/Chicago` time zone.

USA

OSX

Introduction

This page shows the trading hours, holidays, and time zone of the OSX market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the OSX market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:00:00
Tuesday	09:30:00 to 16:00:00
Wednesday	09:30:00 to 16:00:00
Thursday	09:30:00 to 16:00:00
Friday	09:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The OSX market trades in the `America/New York` time zone.

USA

OVX

Introduction

This page shows the trading hours, holidays, and time zone of the OVX market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the OVX market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The OVX market trades in the `America/Chicago` time zone.

USA

PUT

Introduction

This page shows the trading hours, holidays, and time zone of the PUT market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the PUT market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The PUT market trades in the `America/Chicago` time zone.

USA

RLS

Introduction

This page shows the trading hours, holidays, and time zone of the RLS market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the RLS market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The RLS market trades in the `America/Chicago` time zone.

USA

RUA

Introduction

This page shows the trading hours, holidays, and time zone of the RUA market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the RUA market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The RUA market trades in the `America/Chicago` time zone.

USA

RUI

Introduction

This page shows the trading hours, holidays, and time zone of the RUI market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the RUI market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The RUI market trades in the `America/Chicago` time zone.

USA

RUT

Introduction

This page shows the trading hours, holidays, and time zone of the Russell 2000 Index market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Russell 2000 Index market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Russell 2000 Index market trades in the `America/Chicago` time zone.

USA

RVX

Introduction

This page shows the trading hours, holidays, and time zone of the RVX market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the RVX market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The RVX market trades in the `America/Chicago` time zone.

USA

SET

Introduction

This page shows the trading hours, holidays, and time zone of the SET market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the SET market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The SET market trades in the `America/Chicago` time zone.

USA

SGX

Introduction

This page shows the trading hours, holidays, and time zone of the SGX market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the SGX market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The SGX market trades in the `America/Chicago` time zone.

USA

SKEW

Introduction

This page shows the trading hours, holidays, and time zone of the SKEW market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the SKEW market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The SKEW market trades in the `America/Chicago` time zone.

USA

SMLG

Introduction

This page shows the trading hours, holidays, and time zone of the SMLG market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the SMLG market:

Weekday	**Time (America/Los Angeles)**
Monday	06:30:00 to 13:00:00
Tuesday	06:30:00 to 13:00:00
Wednesday	06:30:00 to 13:00:00
Thursday	06:30:00 to 13:00:00
Friday	06:30:00 to 13:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The SMLG market trades in the `America/Los Angeles` time zone.

USA

SP600

Introduction

This page shows the trading hours, holidays, and time zone of the SP600 market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the SP600 market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The SP600 market trades in the `America/Chicago` time zone.

USA

SPGSCI

Introduction

This page shows the trading hours, holidays, and time zone of the SPGSCI market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the SPGSCI market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The SPGSCI market trades in the `America/Chicago` time zone.

USA

SPSIBI

Introduction

This page shows the trading hours, holidays, and time zone of the SPSIBI market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the SPSIBI market:

Weekday	**Time (America/Los Angeles)**
Monday	06:30:00 to 13:00:00
Tuesday	06:30:00 to 13:00:00
Wednesday	06:30:00 to 13:00:00
Thursday	06:30:00 to 13:00:00
Friday	06:30:00 to 13:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The SPSIBI market trades in the `America/Los Angeles` time zone.

USA

SPSV

Introduction

This page shows the trading hours, holidays, and time zone of the SPSV market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the SPSV market:

Weekday	**Time (America/Los Angeles)**
Monday	06:30:00 to 13:00:00
Tuesday	06:30:00 to 13:00:00
Wednesday	06:30:00 to 13:00:00
Thursday	06:30:00 to 13:00:00
Friday	06:30:00 to 13:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The SPSV market trades in the `America/Los Angeles` time zone.

USA

SPX

Introduction

This page shows the trading hours, holidays, and time zone of the S&P 500 Index market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the S&P 500 Index market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:15:00
Tuesday	08:30:00 to 15:15:00
Wednesday	08:30:00 to 15:15:00
Thursday	08:30:00 to 15:15:00
Friday	08:30:00 to 15:15:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The S&P 500 Index market trades in the `America/Chicago` time zone.

USA

SVX

Introduction

This page shows the trading hours, holidays, and time zone of the SVX market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the SVX market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The SVX market trades in the `America/Chicago` time zone.

USA

TNX

Introduction

This page shows the trading hours, holidays, and time zone of the TNX market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the TNX market:

Weekday	**Time (America/Chicago)**
Monday	07:20:00 to 14:00:00
Tuesday	07:20:00 to 14:00:00
Wednesday	07:20:00 to 14:00:00
Thursday	07:20:00 to 14:00:00
Friday	07:20:00 to 14:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The TNX market trades in the `America/Chicago` time zone.

USA

TYX

Introduction

This page shows the trading hours, holidays, and time zone of the TYX market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the TYX market:

Weekday	**Time (America/Chicago)**
Monday	07:20:00 to 14:00:00
Tuesday	07:20:00 to 14:00:00
Wednesday	07:20:00 to 14:00:00
Thursday	07:20:00 to 14:00:00
Friday	07:20:00 to 14:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The TYX market trades in the `America/Chicago` time zone.

USA

UKX

Introduction

This page shows the trading hours, holidays, and time zone of the UKX market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the UKX market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:00:00
Tuesday	09:30:00 to 16:00:00
Wednesday	09:30:00 to 16:00:00
Thursday	09:30:00 to 16:00:00
Friday	09:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The UKX market trades in the `America/New York` time zone.

USA

UTIL

Introduction

This page shows the trading hours, holidays, and time zone of the UTIL market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the UTIL market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The UTIL market trades in the `America/Chicago` time zone.

USA

UTY

Introduction

This page shows the trading hours, holidays, and time zone of the UTY market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the UTY market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:00:00
Tuesday	09:30:00 to 16:00:00
Wednesday	09:30:00 to 16:00:00
Thursday	09:30:00 to 16:00:00
Friday	09:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The UTY market trades in the `America/New York` time zone.

USA

VAF

Introduction

This page shows the trading hours, holidays, and time zone of the VAF market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VAF market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VAF market trades in the `America/Chicago` time zone.

USA

VIF

Introduction

This page shows the trading hours, holidays, and time zone of the VIF market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VIF market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:15:00
Tuesday	08:30:00 to 15:15:00
Wednesday	08:30:00 to 15:15:00
Thursday	08:30:00 to 15:15:00
Friday	08:30:00 to 15:15:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VIF market trades in the `America/Chicago` time zone.

USA

VIN

Introduction

This page shows the trading hours, holidays, and time zone of the VIN market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VIN market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:15:00
Tuesday	08:30:00 to 15:15:00
Wednesday	08:30:00 to 15:15:00
Thursday	08:30:00 to 15:15:00
Friday	08:30:00 to 15:15:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VIN market trades in the `America/Chicago` time zone.

USA

VIX

Introduction

This page shows the trading hours, holidays, and time zone of the CBOE Volatility Index market.

Pre-market Hours

The following table shows the pre-market hours for the CBOE Volatility Index market:

Weekday	**Time (America/Chicago)**
Monday	02:15:00 to 08:15:00
Tuesday	02:15:00 to 08:15:00
Wednesday	02:15:00 to 08:15:00
Thursday	02:15:00 to 08:15:00
Friday	02:15:00 to 08:15:00

Regular Trading Hours

The following table shows the regular trading hours for the CBOE Volatility Index market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:15:00
Tuesday	08:30:00 to 15:15:00
Wednesday	08:30:00 to 15:15:00
Thursday	08:30:00 to 15:15:00
Friday	08:30:00 to 15:15:00

Post-market Hours

The following table shows the post-market hours for the CBOE Volatility Index market:

Weekday	Time (America/Chicago)
Monday	15:15:00 to 16:00:00
Tuesday	15:15:00 to 16:00:00
Wednesday	15:15:00 to 16:00:00
Thursday	15:15:00 to 16:00:00
Friday	15:15:00 to 16:00:00

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01

2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16

2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CBOE Volatility Index market trades in the `America/Chicago` time zone.

USA

VIX1D

Introduction

This page shows the trading hours, holidays, and time zone of the VIX1D market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VIX1D market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VIX1D market trades in the `America/Chicago` time zone.

USA

VIX1Y

Introduction

This page shows the trading hours, holidays, and time zone of the VIX1Y market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VIX1Y market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VIX1Y market trades in the `America/Chicago` time zone.

USA

VIX3M

Introduction

This page shows the trading hours, holidays, and time zone of the VIX3M market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VIX3M market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:15:00
Tuesday	08:30:00 to 15:15:00
Wednesday	08:30:00 to 15:15:00
Thursday	08:30:00 to 15:15:00
Friday	08:30:00 to 15:15:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VIX3M market trades in the `America/Chicago` time zone.

USA

VIX6M

Introduction

This page shows the trading hours, holidays, and time zone of the VIX6M market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VIX6M market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VIX6M market trades in the `America/Chicago` time zone.

USA

VIX9D

Introduction

This page shows the trading hours, holidays, and time zone of the VIX9D market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VIX9D market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VIX9D market trades in the `America/Chicago` time zone.

USA

VOLI

Introduction

This page shows the trading hours, holidays, and time zone of the VOLI market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VOLI market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:00:00
Tuesday	09:30:00 to 16:00:00
Wednesday	09:30:00 to 16:00:00
Thursday	09:30:00 to 16:00:00
Friday	09:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VOLI market trades in the `America/New York` time zone.

USA

VPD

Introduction

This page shows the trading hours, holidays, and time zone of the VPD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VPD market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VPD market trades in the `America/Chicago` time zone.

USA

VPN

Introduction

This page shows the trading hours, holidays, and time zone of the VPN market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VPN market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VPN market trades in the `America/Chicago` time zone.

USA

VRO

Introduction

This page shows the trading hours, holidays, and time zone of the VRO market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VRO market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VRO market trades in the `America/Chicago` time zone.

USA

VVIX

Introduction

This page shows the trading hours, holidays, and time zone of the VVIX market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VVIX market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VVIX market trades in the `America/Chicago` time zone.

USA

VWA

Introduction

This page shows the trading hours, holidays, and time zone of the VWA market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VWA market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VWA market trades in the `America/Chicago` time zone.

USA

VWB

Introduction

This page shows the trading hours, holidays, and time zone of the VWB market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VWB market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VWB market trades in the `America/Chicago` time zone.

USA

VXD

Introduction

This page shows the trading hours, holidays, and time zone of the VXD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VXD market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VXD market trades in the `America/Chicago` time zone.

USA

VXN

Introduction

This page shows the trading hours, holidays, and time zone of the VXN market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VXN market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VXN market trades in the `America/Chicago` time zone.

USA

VXO

Introduction

This page shows the trading hours, holidays, and time zone of the VXO market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VXO market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VXO market trades in the `America/Chicago` time zone.

USA

VXSLV

Introduction

This page shows the trading hours, holidays, and time zone of the VXSLV market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VXSLV market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VXSLV market trades in the `America/Chicago` time zone.

USA

VXTH

Introduction

This page shows the trading hours, holidays, and time zone of the VXTH market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VXTH market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VXTH market trades in the `America/Chicago` time zone.

USA

VXTLT

Introduction

This page shows the trading hours, holidays, and time zone of the VXTLT market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the VXTLT market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The VXTLT market trades in the `America/Chicago` time zone.

USA

XAU

Introduction

This page shows the trading hours, holidays, and time zone of the XAU market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the XAU market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:00:00
Tuesday	09:30:00 to 16:00:00
Wednesday	09:30:00 to 16:00:00
Thursday	09:30:00 to 16:00:00
Friday	09:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The XAU market trades in the `America/New York` time zone.

USA

XDA

Introduction

This page shows the trading hours, holidays, and time zone of the XDA market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the XDA market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:00:00
Tuesday	09:30:00 to 16:00:00
Wednesday	09:30:00 to 16:00:00
Thursday	09:30:00 to 16:00:00
Friday	09:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The XDA market trades in the `America/New York` time zone.

USA

XDB

Introduction

This page shows the trading hours, holidays, and time zone of the XDB market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the XDB market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:00:00
Tuesday	09:30:00 to 16:00:00
Wednesday	09:30:00 to 16:00:00
Thursday	09:30:00 to 16:00:00
Friday	09:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The XDB market trades in the `America/New York` time zone.

USA

XEO

Introduction

This page shows the trading hours, holidays, and time zone of the XEO market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the XEO market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The XEO market trades in the `America/Chicago` time zone.

USA

XMI

Introduction

This page shows the trading hours, holidays, and time zone of the XMI market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the XMI market:

Weekday	**Time (America/Los Angeles)**
Monday	06:30:00 to 13:00:00
Tuesday	06:30:00 to 13:00:00
Wednesday	06:30:00 to 13:00:00
Thursday	06:30:00 to 13:00:00
Friday	06:30:00 to 13:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The XMI market trades in the `America/Los Angeles` time zone.

USA

XNDX

Introduction

This page shows the trading hours, holidays, and time zone of the XNDX market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the XNDX market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:00:00
Tuesday	09:30:00 to 16:00:00
Wednesday	09:30:00 to 16:00:00
Thursday	09:30:00 to 16:00:00
Friday	09:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The XNDX market trades in the `America/New York` time zone.

USA

XSP

Introduction

This page shows the trading hours, holidays, and time zone of the XSP market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the XSP market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:00:00
Tuesday	08:30:00 to 15:00:00
Wednesday	08:30:00 to 15:00:00
Thursday	08:30:00 to 15:00:00
Friday	08:30:00 to 15:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Indices market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-09
2025-01-20	2025-02-17	2025-04-18	2025-05-26	2025-06-19
2025-07-04	2025-09-01	2025-11-27	2025-12-25	2026-01-01
2026-01-19	2026-02-16	2026-04-03	2026-05-25	2026-06-19
2026-07-03	2026-09-07	2026-11-26	2026-12-25	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The XSP market trades in the `America/Chicago` time zone.

Market Hours

EUREX

Introduction

This page shows the trading hours, holidays, and time zone of the EUREX Indices market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUREX Indices market:

Weekday	**Time (Europe/Berlin)**
Monday	09:00:00 to 17:30:00
Tuesday	09:00:00 to 17:30:00
Wednesday	09:00:00 to 17:30:00
Thursday	09:00:00 to 17:30:00
Friday	09:00:00 to 17:30:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the EUREX Indices market:

Date (*yyyy-mm-dd*)				
2000-04-21	2000-04-24	2000-05-01	2000-12-25	2000-12-26
2001-01-01	2001-04-13	2001-04-16	2001-05-01	2001-12-24
2001-12-25	2001-12-26	2001-12-31	2002-01-01	2002-03-29
2002-04-01	2002-05-01	2002-12-24	2002-12-25	2002-12-26
2002-12-31	2003-01-01	2003-04-18	2003-04-21	2004-01-01
2004-04-09	2004-04-12	2004-05-01	2004-12-24	2004-12-25
2004-12-26	2004-12-31	2005-03-25	2005-03-28	2005-12-26
2006-04-14	2006-04-17	2006-05-01	2006-12-25	2006-12-26

2007-01-01	2007-04-06	2007-04-09	2007-05-01	2007-12-24
2007-12-25	2007-12-26	2007-12-31	2008-01-01	2008-03-21
2008-03-24	2008-05-01	2008-12-24	2008-12-25	2008-12-26
2008-12-31	2009-01-01	2009-04-10	2009-04-13	2009-05-01
2009-12-24	2009-12-25	2009-12-31	2010-01-01	2010-04-02
2010-04-05	2010-12-24	2010-12-31	2011-04-22	2011-04-25
2011-12-26	2012-04-06	2012-04-09	2012-05-01	2012-12-24
2012-12-25	2012-12-26	2012-12-31	2013-01-01	2013-03-29
2013-04-01	2013-05-01	2013-12-24	2013-12-25	2013-12-26
2013-12-31	2014-01-01	2014-04-18	2014-04-21	2014-05-01
2014-12-24	2014-12-25	2014-12-26	2014-12-31	2015-01-01
2015-04-03	2015-04-06	2015-05-01	2015-05-25	2015-12-24
2015-12-25	2015-12-26	2015-12-31	2016-01-01	2016-03-25
2016-03-28	2016-12-26	2017-04-14	2017-04-17	2017-05-01
2017-12-25	2017-12-26	2018-01-01	2018-03-30	2018-04-02
2018-05-01	2018-12-24	2018-12-25	2018-12-26	2018-12-31
2019-01-01	2019-04-19	2019-04-22	2019-05-01	2019-12-24
2019-12-25	2019-12-26	2019-12-31	2020-01-01	2020-04-10
2020-04-13	2020-05-01	2020-12-24	2020-12-25	2020-12-31
2021-01-01	2021-04-02	2021-04-05	2021-12-24	2021-12-31
2022-04-15	2022-04-18	2022-12-26	2023-04-07	2023-04-10
2023-05-01	2023-12-25	2023-12-26	2024-01-01	2024-03-29
2024-04-01	2024-05-01	2024-12-24	2024-12-25	2024-12-26
2024-12-31	2025-01-01	2025-04-18	2025-04-21	2025-05-01
2025-12-24	2025-12-25	2025-12-26	2025-12-31	2026-01-01
2026-04-03	2026-04-06	2026-05-01	2026-12-24	2026-12-25
2026-12-31	2027-01-01			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUREX Indices market trades in the `Europe/Berlin` time zone.

Assets With Other Hours

The following table shows the indices that have different trading periods than the overall EUREX Indices market:

Symbol	Name
DAX	DAX

EUREX

DAX

Introduction

This page shows the trading hours, holidays, and time zone of the DAX market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DAX market:

Weekday	**Time (Europe/Berlin)**
Monday	09:00:00 to 17:40:00
Tuesday	09:00:00 to 17:40:00
Wednesday	09:00:00 to 17:40:00
Thursday	09:00:00 to 17:40:00
Friday	09:00:00 to 17:40:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the EUREX Indices market:

Date (*yyyy-mm-dd*)				
2000-04-21	2000-04-24	2000-05-01	2000-12-25	2000-12-26
2001-01-01	2001-04-13	2001-04-16	2001-05-01	2001-12-24
2001-12-25	2001-12-26	2001-12-31	2002-01-01	2002-03-29
2002-04-01	2002-05-01	2002-12-24	2002-12-25	2002-12-26
2002-12-31	2003-01-01	2003-04-18	2003-04-21	2004-01-01
2004-04-09	2004-04-12	2004-05-01	2004-12-24	2004-12-25
2004-12-26	2004-12-31	2005-03-25	2005-03-28	2005-12-26
2006-04-14	2006-04-17	2006-05-01	2006-12-25	2006-12-26

2007-01-01	2007-04-06	2007-04-09	2007-05-01	2007-12-24
2007-12-25	2007-12-26	2007-12-31	2008-01-01	2008-03-21
2008-03-24	2008-05-01	2008-12-24	2008-12-25	2008-12-26
2008-12-31	2009-01-01	2009-04-10	2009-04-13	2009-05-01
2009-12-24	2009-12-25	2009-12-31	2010-01-01	2010-04-02
2010-04-05	2010-12-24	2010-12-31	2011-04-22	2011-04-25
2011-12-26	2012-04-06	2012-04-09	2012-05-01	2012-12-24
2012-12-25	2012-12-26	2012-12-31	2013-01-01	2013-03-29
2013-04-01	2013-05-01	2013-12-24	2013-12-25	2013-12-26
2013-12-31	2014-01-01	2014-04-18	2014-04-21	2014-05-01
2014-12-24	2014-12-25	2014-12-26	2014-12-31	2015-01-01
2015-04-03	2015-04-06	2015-05-01	2015-05-25	2015-12-24
2015-12-25	2015-12-26	2015-12-31	2016-01-01	2016-03-25
2016-03-28	2016-12-26	2017-04-14	2017-04-17	2017-05-01
2017-12-25	2017-12-26	2018-01-01	2018-03-30	2018-04-02
2018-05-01	2018-12-24	2018-12-25	2018-12-26	2018-12-31
2019-01-01	2019-04-19	2019-04-22	2019-05-01	2019-12-24
2019-12-25	2019-12-26	2019-12-31	2020-01-01	2020-04-10
2020-04-13	2020-05-01	2020-12-24	2020-12-25	2020-12-31
2021-01-01	2021-04-02	2021-04-05	2021-12-24	2021-12-31
2022-04-15	2022-04-18	2022-12-26	2023-04-07	2023-04-10
2023-05-01	2023-12-25	2023-12-26	2024-01-01	2024-03-29
2024-04-01	2024-05-01	2024-12-24	2024-12-25	2024-12-26
2024-12-31	2025-01-01	2025-04-18	2025-04-21	2025-05-01
2025-12-24	2025-12-25	2025-12-26	2025-12-31	2026-01-01
2026-04-03	2026-04-06	2026-05-01	2026-12-24	2026-12-25
2026-12-31	2027-01-01			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DAX market trades in the `Europe/Berlin` time zone.

Market Hours

HKFE

Introduction

This page shows the trading hours, holidays, and time zone of the HKFE Indices market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the HKFE Indices market:

Weekday	**Time (Asia/Hong Kong)**
Monday	09:30:00 to 12:00:00, 13:00:00 to 16:00:00
Tuesday	09:30:00 to 12:00:00, 13:00:00 to 16:00:00
Wednesday	09:30:00 to 12:00:00, 13:00:00 to 16:00:00
Thursday	09:30:00 to 12:00:00, 13:00:00 to 16:00:00
Friday	09:30:00 to 12:00:00, 13:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the HKFE Indices market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-30	2017-01-31	2017-04-04	2017-04-14
2017-04-17	2017-05-01	2017-05-03	2017-05-30	2017-10-02
2017-10-05	2017-12-25	2017-12-26	2018-01-01	2018-02-16
2018-02-19	2018-03-30	2018-04-02	2018-04-05	2018-05-01
2018-05-22	2018-06-18	2018-07-02	2018-09-25	2018-10-01
2018-10-17	2018-12-25	2018-12-26	2019-01-01	2019-02-05
2019-02-06	2019-02-07	2019-04-05	2019-04-19	2019-04-22
2019-05-01	2019-05-13	2019-06-07	2019-07-01	2019-10-01

2019-10-07	2019-12-25	2019-12-26	2020-01-01	2020-01-27
2020-01-28	2020-04-10	2020-04-13	2020-04-30	2020-05-01
2020-06-25	2020-07-01	2020-10-01	2020-10-02	2020-10-26
2020-12-25	2021-01-01	2021-02-12	2021-02-15	2021-04-02
2021-04-05	2021-04-06	2021-05-19	2021-06-14	2021-07-01
2021-09-22	2021-10-01	2021-10-14	2021-12-27	2022-02-01
2022-02-02	2022-02-03	2022-04-05	2022-04-15	2022-04-18
2022-05-02	2022-05-09	2022-06-03	2022-07-01	2022-09-12
2022-10-04	2022-12-26	2022-12-27	2023-01-02	2023-01-23
2023-01-24	2023-01-25	2023-04-05	2023-04-07	2023-04-10
2023-05-01	2023-05-26	2023-06-22	2023-10-02	2023-10-23
2023-12-25	2023-12-26	2024-01-01	2024-02-12	2024-02-13
2024-03-29	2024-04-01	2024-04-04	2024-05-01	2024-05-15
2024-06-10	2024-07-01	2024-09-18	2024-10-01	2024-10-11
2024-12-25	2024-12-26	2025-01-01	2025-01-29	2025-01-30
2025-01-31	2025-04-04	2025-04-18	2025-04-21	2025-05-01
2025-05-05	2025-07-01	2025-10-01	2025-10-07	2025-10-29
2025-12-25	2025-12-26	2026-01-01	2026-02-17	2026-02-18
2026-02-19	2026-04-03	2026-04-06	2026-04-07	2026-05-01
2026-05-25	2026-06-19	2026-07-01	2026-10-01	2026-10-19
2026-12-25				

Early Closes

The following table shows the early closes for the HKFE Indices market:

Date (*yyyy-mm-dd*)	Time Of Market Close (Asia/Hong Kong)
2018-02-15	12:00:00
2018-12-24	12:00:00
2018-12-31	12:00:00
2019-02-04	12:00:00
2019-12-24	12:00:00
2019-12-31	12:00:00
2020-01-24	12:00:00
2020-12-24	12:00:00
2020-12-31	12:00:00
2021-02-11	12:00:00
2021-12-24	12:00:00
2021-12-31	12:00:00
2022-01-31	12:00:00
2024-02-09	12:00:00
2024-12-24	12:00:00
2024-12-31	12:00:00
2025-01-28	12:00:00
2025-12-24	12:00:00
2025-12-31	12:00:00
2026-02-16	12:00:00
2026-12-24	12:00:00
2026-12-31	12:00:00

Late Opens

There are no days with late opens.

Time Zone

The HKFE Indices market trades in the `Asia/Hong Kong` time zone.

Assets With Other Hours

The following table shows the indices that have different trading periods than the overall HKFE Indices market:

Symbol	Name
HSI	Hang Seng Index

HKFE

HSI

Introduction

This page shows the trading hours, holidays, and time zone of the Hang Seng Index market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Hang Seng Index market:

Weekday	**Time (Asia/Hong Kong)**
Monday	09:30:00 to 12:00:00, 13:00:00 to 16:00:00
Tuesday	09:30:00 to 12:00:00, 13:00:00 to 16:00:00
Wednesday	09:30:00 to 12:00:00, 13:00:00 to 16:00:00
Thursday	09:30:00 to 12:00:00, 13:00:00 to 16:00:00
Friday	09:30:00 to 12:00:00, 13:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the HKFE Indices market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-30	2017-01-31	2017-04-04	2017-04-14
2017-04-17	2017-05-01	2017-05-03	2017-05-30	2017-10-02
2017-10-05	2017-12-25	2017-12-26	2018-01-01	2018-02-16
2018-02-19	2018-03-30	2018-04-02	2018-04-05	2018-05-01
2018-05-22	2018-06-18	2018-07-02	2018-09-25	2018-10-01
2018-10-17	2018-12-25	2018-12-26	2019-01-01	2019-02-05
2019-02-06	2019-02-07	2019-04-05	2019-04-19	2019-04-22
2019-05-01	2019-05-13	2019-06-07	2019-07-01	2019-10-01

2019-10-07	2019-12-25	2019-12-26	2020-01-01	2020-01-27
2020-01-28	2020-04-10	2020-04-13	2020-04-30	2020-05-01
2020-06-25	2020-07-01	2020-10-01	2020-10-02	2020-10-26
2020-12-25	2021-01-01	2021-02-12	2021-02-15	2021-04-02
2021-04-05	2021-04-06	2021-05-19	2021-06-14	2021-07-01
2021-09-22	2021-10-01	2021-10-14	2021-12-27	2022-02-01
2022-02-02	2022-02-03	2022-04-05	2022-04-15	2022-04-18
2022-05-02	2022-05-09	2022-06-03	2022-07-01	2022-09-12
2022-10-04	2022-12-26	2022-12-27	2023-01-02	2023-01-23
2023-01-24	2023-01-25	2023-04-05	2023-04-07	2023-04-10
2023-05-01	2023-05-26	2023-06-22	2023-10-02	2023-10-23
2023-12-25	2023-12-26	2024-01-01	2024-02-12	2024-02-13
2024-03-29	2024-04-01	2024-04-04	2024-05-01	2024-05-15
2024-06-10	2024-07-01	2024-09-18	2024-10-01	2024-10-11
2024-12-25	2024-12-26	2025-01-01	2025-01-29	2025-01-30
2025-01-31	2025-04-04	2025-04-18	2025-04-21	2025-05-01
2025-05-05	2025-07-01	2025-10-01	2025-10-07	2025-10-29
2025-12-25	2025-12-26	2026-01-01	2026-02-17	2026-02-18
2026-02-19	2026-04-03	2026-04-06	2026-04-07	2026-05-01
2026-05-25	2026-06-19	2026-07-01	2026-10-01	2026-10-19
2026-12-25				

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Hang Seng Index market trades in the `Asia/Hong Kong` time zone.

Market Hours

OSE

Introduction

This page shows the trading hours, holidays, and time zone of the Japan Indices market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Japan Indices market:

Weekday	**Time (Asia/Tokyo)**
Monday	09:00:00 to 11:30:00, 12:30:00 to 15:30:00
Tuesday	09:00:00 to 11:30:00, 12:30:00 to 15:30:00
Wednesday	09:00:00 to 11:30:00, 12:30:00 to 15:30:00
Thursday	09:00:00 to 11:30:00, 12:30:00 to 15:30:00
Friday	09:00:00 to 11:30:00, 12:30:00 to 15:30:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Japan Indices market:

Date (*yyyy-mm-dd*)				
2016-01-01	2016-01-11	2016-02-11	2016-03-21	2016-04-29
2016-05-03	2016-05-04	2016-05-05	2016-07-18	2016-08-11
2016-09-19	2016-09-22	2016-10-10	2016-11-03	2016-11-23
2016-12-23	2016-12-31	2017-01-02	2017-01-03	2017-01-09
2017-03-20	2017-05-03	2017-05-04	2017-05-05	2017-07-17
2017-08-11	2017-09-18	2017-10-09	2017-11-03	2017-11-23
2018-01-01	2018-01-02	2018-01-03	2018-01-08	2018-02-12
2018-03-21	2018-04-30	2018-05-03	2018-05-04	2018-07-16

2018-09-17	2018-09-24	2018-10-08	2018-11-23	2018-12-24
2018-12-31	2019-01-01	2019-01-02	2019-01-03	2019-01-14
2019-02-11	2019-03-21	2019-04-29	2019-04-30	2019-05-01
2019-05-02	2019-05-03	2019-05-06	2019-07-15	2019-08-12
2019-09-16	2019-09-23	2019-10-14	2019-10-22	2019-11-04
2019-12-31	2020-01-01	2020-01-02	2020-01-03	2020-01-13
2020-02-11	2020-02-24	2020-03-20	2020-04-29	2020-05-04
2020-05-05	2020-05-06	2020-07-23	2020-07-24	2020-08-10
2020-09-21	2020-09-22	2020-11-03	2020-11-23	2020-12-31
2021-01-01	2021-01-11	2021-02-11	2021-02-23	2021-04-29
2021-05-03	2021-05-04	2021-05-05	2021-07-22	2021-07-23
2021-08-09	2021-09-20	2021-09-23	2021-11-03	2021-11-23
2021-12-31	2022-01-03	2022-01-10	2022-02-11	2022-02-23
2022-03-21	2022-04-29	2022-05-03	2022-05-04	2022-05-05
2022-07-18	2022-08-11	2022-09-19	2022-10-10	2022-11-03
2022-11-23	2023-01-02	2023-01-03	2023-01-09	2023-02-23
2023-03-21	2023-05-03	2023-05-04	2023-05-05	2023-07-17
2023-08-11	2023-09-18	2023-10-09	2023-11-03	2023-11-23
2024-01-01	2024-01-02	2024-01-03	2024-01-08	2024-02-12
2024-02-23	2024-03-20	2024-04-29	2024-05-03	2024-05-06
2024-07-15	2024-08-12	2024-09-16	2024-09-23	2024-10-14
2024-11-04	2024-12-31	2025-01-01	2025-01-02	2025-01-03
2025-01-13	2025-02-11	2025-02-24	2025-03-20	2025-04-29
2025-05-05	2025-05-06	2025-07-21	2025-08-11	2025-09-15
2025-09-23	2025-10-13	2025-11-03	2025-11-24	2025-12-31
2026-01-01	2026-01-02	2026-01-03	2026-01-12	2026-02-11
2026-02-23	2026-03-20	2026-04-29	2026-05-03	2026-05-04
2026-05-05	2026-05-06	2026-07-20	2026-08-11	2026-09-21

2026-09-22	2026-09-23	2026-10-12	2026-11-03	2026-11-23
2026-12-31				

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Japan Indices market trades in the `Asia/Tokyo` time zone.

Assets With Other Hours

The following table shows the indices that have different trading periods than the overall Japan Indices market:

Symbol	**Name**
N225	N225

OSE

N225

Introduction

This page shows the trading hours, holidays, and time zone of the N225 market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the N225 market:

Weekday	**Time (Asia/Tokyo)**
Monday	09:00:00 to 11:30:00, 12:30:00 to 15:30:00
Tuesday	09:00:00 to 11:30:00, 12:30:00 to 15:30:00
Wednesday	09:00:00 to 11:30:00, 12:30:00 to 15:30:00
Thursday	09:00:00 to 11:30:00, 12:30:00 to 15:30:00
Friday	09:00:00 to 11:30:00, 12:30:00 to 15:30:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Japan Indices market:

Date (*yyyy-mm-dd*)				
2016-01-01	2016-01-11	2016-02-11	2016-03-21	2016-04-29
2016-05-03	2016-05-04	2016-05-05	2016-07-18	2016-08-11
2016-09-19	2016-09-22	2016-10-10	2016-11-03	2016-11-23
2016-12-23	2016-12-31	2017-01-02	2017-01-03	2017-01-09
2017-03-20	2017-05-03	2017-05-04	2017-05-05	2017-07-17
2017-08-11	2017-09-18	2017-10-09	2017-11-03	2017-11-23
2018-01-01	2018-01-02	2018-01-03	2018-01-08	2018-02-12
2018-03-21	2018-04-30	2018-05-03	2018-05-04	2018-07-16

2018-09-17	2018-09-24	2018-10-08	2018-11-23	2018-12-24
2018-12-31	2019-01-01	2019-01-02	2019-01-03	2019-01-14
2019-02-11	2019-03-21	2019-04-29	2019-04-30	2019-05-01
2019-05-02	2019-05-03	2019-05-06	2019-07-15	2019-08-12
2019-09-16	2019-09-23	2019-10-14	2019-10-22	2019-11-04
2019-12-31	2020-01-01	2020-01-02	2020-01-03	2020-01-13
2020-02-11	2020-02-24	2020-03-20	2020-04-29	2020-05-04
2020-05-05	2020-05-06	2020-07-23	2020-07-24	2020-08-10
2020-09-21	2020-09-22	2020-11-03	2020-11-23	2020-12-31
2021-01-01	2021-01-11	2021-02-11	2021-02-23	2021-04-29
2021-05-03	2021-05-04	2021-05-05	2021-07-22	2021-07-23
2021-08-09	2021-09-20	2021-09-23	2021-11-03	2021-11-23
2021-12-31	2022-01-03	2022-01-10	2022-02-11	2022-02-23
2022-03-21	2022-04-29	2022-05-03	2022-05-04	2022-05-05
2022-07-18	2022-08-11	2022-09-19	2022-10-10	2022-11-03
2022-11-23	2023-01-02	2023-01-03	2023-01-09	2023-02-23
2023-03-21	2023-05-03	2023-05-04	2023-05-05	2023-07-17
2023-08-11	2023-09-18	2023-10-09	2023-11-03	2023-11-23
2024-01-01	2024-01-02	2024-01-03	2024-01-08	2024-02-12
2024-02-23	2024-03-20	2024-04-29	2024-05-03	2024-05-06
2024-07-15	2024-08-12	2024-09-16	2024-09-23	2024-10-14
2024-11-04	2024-12-31	2025-01-01	2025-01-02	2025-01-03
2025-01-13	2025-02-11	2025-02-24	2025-03-20	2025-04-29
2025-05-05	2025-05-06	2025-07-21	2025-08-11	2025-09-15
2025-09-23	2025-10-13	2025-11-03	2025-11-24	2025-12-31
2026-01-01	2026-01-02	2026-01-03	2026-01-12	2026-02-11
2026-02-23	2026-03-20	2026-04-29	2026-05-03	2026-05-04
2026-05-05	2026-05-06	2026-07-20	2026-08-11	2026-09-21

2026-09-22	2026-09-23	2026-10-12	2026-11-03	2026-11-23
2026-12-31				

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The N225 market trades in the `Asia/Tokyo` time zone.

Asset Classes

Index Options

Index Options are a financial derivative that gives the holder the right (but not the obligation) to buy or sell the value of an underlying Index, such as the S&P 500 index, at the stated exercise price. No actual assets are bought or sold.

Requesting Data

Handling Data

Greeks and Implied Volatility

Market Hours

See Also

BasicTemplateIndexOptionsAlgorithm.py
BasicTemplateIndexOptionsAlgorithm.cs

Index Options

Requesting Data

Request Index Options data in your algorithm to receive a feed of contract prices in the `OnData` method. For more information about the specific dataset we use for backtests, see the US Index Options dataset listing . To trade Index Options live, you can use one of the brokerage data providers . Index Options are exclusively European-style Options.

To request Index Options data, call one of the following methods:

- `AddIndexOption` to add a basket of contracts for a single underlying Index
- `AddIndexOptionContract` to add individual contracts for an underlying Index

Request and Filter Contracts for Option Strategies

`AddIndexOption`

Request and Filter All Contracts

`AddIndexOption`

Select Individual Contracts

`AddIndexOptionContract`

See Also

Introduction to Options
Applied Options

Requesting Data

Universes

Introduction

The `AddIndexOption` method enables you to select a basket of Option contracts for an underlying Index. To form your universe of contracts, you can filter them down by their strike price and expiry or you can choose to a subset of the contracts that form popular Option strategies . If you want to subscribe to individual contracts one-by-one instead of a set of contracts, see Individual Contracts .

Create Universes

To add a universe of Index Option contracts, in the `Initialize` method, call the `AddIndexOption` method. This method returns an `Option` object, which has a `SetFilter` method you can call to filter the set of tradable contract down to just the contracts you want.

```C#
public class BasicIndexOptionAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        UniverseSettings.Asynchronous = true;
        var option = AddIndexOption("SPX", "SPXW");
        option.SetFilter(Filter);
        _symbol = option.Symbol;
    }

    private OptionFilterUniverse Filter(OptionFilterUniverse universe)
    {
        return universe.IncludeWeeklys().Expiration(0, 7).Delta(0.35m, 0.75m);
    }

    public override void OnData(Slice data)
    {
        if (data.OptionChains.TryGetValue(_symbol, out var chain))
        {
            var contract = chain.OrderBy(x => x.Expiry).ThenBy(x => x.Greeks.Delta).FirstOrDefault();
        }
    }
}
```

For more information about the `AddIndexOption` method, see Create Universes .

Filter by Investment Strategy

Options trading strategies consist of simultaneously buying or selling one or more Option contracts of the same underlying Index with different features. The following table describes the filter methods of the `OptionFilterUniverse` class that select contract for Option strategies.

The following table describes the filter methods of the `OptionFilterUniverse` class that select contracts for Option strategies:

`NakedCall(int minDaysTillExpiry, decimal strikeFromAtm)` Selects a call contract to form Naked Call , Covered Call , or Protective Call Option strategies.

`NakedPut(int minDaysTillExpiry, decimal strikeFromAtm)`

Selects a put contract to form Naked Put , Covered Put , or Protective Put Option strategies.

`CallSpread(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal lowerStrikeFromAtm)`

Selects two call contracts to form Bull Call Spread or Bear Call Spread Option strategies.

`PutSpread(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal lowerStrikeFromAtm)`

Selects two put contracts to form Bull Put Spread or Bear Put Spread Option strategies.

`CallCalendarSpread(decimal strikeFromAtm, int minNearDaysTillExpiry, int minFarDaysTillExpiry)`

Selects two call contracts to form Long Call Calendar Spread or Short Call Calendar Spread Option strategies.

`PutCalendarSpread(decimal strikeFromAtm, int minNearDaysTillExpiry, int minFarDaysTillExpiry)`

Selects two put contracts to form Long Put Calendar Spread or Short Put Calendar Spread Option strategies.

`Strangle(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal lowerStrikeFromAtm)`

Selects two contracts to form Long Strangle or Short Strangle Option strategies.

`Straddle(int minDaysTillExpiry)`

Selects two contracts to form Long Straddle or Short Straddle Option strategies.

`ProtectiveCollar(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal lowerStrikeFromAtm)`

Selects two contracts to form Protective Collar Option strategies.

`Conversion(int minDaysTillExpiry, decimal strikeFromAtm)`

Selects two contracts to form Conversion or Reverse Conversion Option strategies.

`CallButterfly(int minDaysTillExpiry, decimal strikeSpread)`

Selects three contracts to form Long Call Butterfly or Short Call Butterfly Option strategies.

`PutButterfly(int minDaysTillExpiry, decimal strikeSpread)`

Selects three contracts to form Long Put Butterfly or Short Put Butterfly Option strategies.

`IronButterfly(int minDaysTillExpiry, decimal strikeSpread)`

Selects four contracts to form Long Iron Butterfly or Short Iron Butterfly Option strategies.

`IronCondor(int minDaysTillExpiry, decimal nearStrikeSpread, decimal farStrikeSpread)` Selects four contracts to form Long Iron Condor or Short Iron Condor Option strategies.
`BoxSpread(int minDaysTillExpiry, decimal strikeSpread)` Selects four contracts to form Box Spread or Short Box Spread Option strategies.
`JellyRoll(decimal strikeFromAtm, int minNearDaysTillExpiry, int minFarDaysTillExpiry)` Selects four contracts to form Jelly Roll or Short Jelly Roll Option strategies.
`CallLadder(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal middleStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects four contracts to form Bear Call Ladder or Bull Call Ladder Option strategies.
`PutLadder(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal middleStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects four contracts to form Bear Put Ladder or Bull Put Ladder Option strategies.

The preceding methods return an `OptionFilterUniverse`, so you can chain the methods together.

```C#
// Example 1: Select a Straddle
option.SetFilter(optionFilterUniverse => optionFilterUniverse.Straddle(30, 5, 10));

// Example 2: Select the contracts (including weeklys) that expire in the next 30 days that form an Iron Condor
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().Strikes(-20, 20).Expiration(0, 30).IronCondor(30, 5, 10));

// Example 3: Select the 0DTE contracts that form an Strangle
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().Expiration(0, 0).Strangle(30, 5, -10));
```

Filter by Implied Volatility and Greeks

Option price models compute the theoretical price of Option contracts, their implied volatility, and their Greek values. Theoretical prices can help you detect undervalued and overvalued contracts, implied volatility can provide you insight into the upcoming volatility of the underlying security, and Greek values can help you hedge your portfolio. The following table describes the filter methods of the `OptionFilterUniverse` class that select contract for a range of implied volatility, Greek values, and open interest.

The following table describes the filter methods of the `OptionFilterUniverse` class that select contract for a range of implied volatility, Greek values, and open interest:

`IV(decimal min, decimal max)`
`ImpliedVolatility(decimal min, decimal max)`
 	
Selects a contract with implied volatility within the range you set.

`D(decimal min, decimal max)`
`Delta(decimal min, decimal max)`
 	
Selects a contract with delta within the range you set.

`G(decimal min, decimal max)`
`Gamma(decimal min, decimal max)`
 	
Selects a contract with gamma within the range you set.

`R(decimal min, decimal max)`
`Rho(decimal min, decimal max)`
 	
Selects a contract with rho within the range you set.

`V(decimal min, decimal max)`
`Vega(decimal min, decimal max)`
 	
Selects a contract with vega within the range you set.

`T(decimal min, decimal max)`
`Theta(decimal min, decimal max)`
 	
Selects a contract with theta within the range you set.

`OI(long min, long max)`
`OpenInterest(decimal min, decimal max)`
 	
Selects a contract with open interest within the range you set.

The preceding methods return an `OptionFilterUniverse` , so you can chain the methods together.

```
C#
```

```
// Example 1: Select the contracts with delta between 0.25 and 0.75
option.SetFilter(optionFilterUniverse => optionFilterUniverse.Delta(0.25m, 0.75m));

// Example 2: Select the contracts (including weeklys) that expire in the next 90 days with implied volatility below
20%
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().IV(0, 20).Expiration(0, 90));

// Example 3: Select the contracts (including weeklys) that expire in the next 30 days with implied volatility below
20% that forms an Iron Condor
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().IV(0, 20).Expiration(0,
30).IronCondor(30, 5, 10));
```

Filter by Other Contract Properties

To set a contract filter, in the `Initialize` method, call the `SetFilter` method of the `Option` object.

```C#
// Select contracts that have a strike price within 1 strike level above and below the underlying price
option.SetFilter(minStrike: -1, maxStrike: 1);

// Select contracts that expire within 30 days
option.SetFilter(minExpiry: TimeSpan.FromDays(0), maxExpiry: TimeSpan.FromDays(30));

// Select contracts that have a strike price within 1 strike level and expire within 30 days
option.SetFilter(minStrike: -1, maxStrike: 1, minExpiry: TimeSpan.FromDays(0), maxExpiry: TimeSpan.FromDays(30));

// Select call contracts
option.SetFilter(optionFilterUniverse => optionFilterUniverse.CallsOnly());
```

The following table describes the available filter techniques:

`SetFilter(int minStrike, int maxStrike)` Selects the contracts that have a strike price within a minimum and maximum strike level relative to the underlying price. For example, say the underlying price is 302*andthereare*strikes*at*every5. If you set `minStrike` to -1 and `maxStrike` to 1, LEAN selects the contracts that have a strike of 300*or*305. This filter runs asynchronously by default.
`SetFilter(TimeSpan minExpiry, TimeSpan maxExpiry)` Selects the contracts that expire within the range you set. This filter runs asynchronously by default.
`SetFilter(int minStrike, int maxStrike, TimeSpan minExpiry, TimeSpan maxExpiry)` Selects the contracts that expire and have a strike within the range you set. This filter runs asynchronously by default.
`SetFilter(Func<OptionFilterUniverse, OptionFilterUniverse> universeFunc)` Selects the contracts that a function selects.

The following table describes the filter methods of the `OptionFilterUniverse` class:

`Strikes(int minStrike, int maxStrike)` Selects contracts that are within `minStrike` strikes below the underlying price and `maxStrike` strikes above the underlying price.
`CallsOnly()` Selects call contracts.
`PutsOnly()` Selects put contracts.
`StandardsOnly()` Selects standard contracts.
`IncludeWeeklys()` Selects non-standard weeklys contracts.
`WeeklysOnly()` Selects weekly contracts.
`FrontMonth()` Selects the front month contract.
`BackMonths()` Selects the non-front month contracts.
`BackMonth()` Selects the back month contracts.
`Expiration(int minExpiryDays, int maxExpiryDays)` Selects contracts that expire within a range of dates relative to the current day.
`Contracts(IEnumerable<Symbol> contracts)` Selects a list of contracts.
`Contracts(Func<IEnumerable<Symbol>, IEnumerable< Symbol>> contractSelector)` Selects contracts that a selector function selects.

The preceding methods return an `OptionFilterUniverse` , so you can chain the methods together.

```C#
// Example 1: Select the front month call contracts
option.SetFilter(optionFilterUniverse => optionFilterUniverse.CallsOnly().FrontMonth());

// Example 2: Select the contracts (including weeklys) that expire in the next 90 days
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().Strikes(-20, 20).Expiration(0, 90));

// Example 3: Select the contracts (including weeklys) that expire in the next 30 days that form an Iron Condor
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().Strikes(-20, 20).Expiration(0,
30).IronCondor(30, 5, 10));
```

Some of the preceding filter methods only set an internal enumeration in the `OptionFilterUniverse` that it uses later on in the filter process. This subset of filter methods don't immediately reduce the number of contract `Symbol` objects in the `OptionFilterUniverse` .

Default Filter

By default, LEAN subscribes to the Option contracts that have the following characteristics:

- Standard type (exclude weeklys)
- Within 1 strike price of the underlying asset price
- Expire within 35 days

To adjust the universe of contracts, set a filter. The filter usually runs at the first bar of every day. When the filter selects a contract that isn't currently in your universe, LEAN adds the new contract data to the next `Slice` that it passes to the `OnData` method.

Historical Data

To get historical chains for an Index Option, see Historical Data .

Examples

The following examples demonstrate some common practices for requesting Index Options universe data.

Example 1: Selecting for 0DTE Contracts

0DTE Options are option contracts that expire on the same day they are traded. To create a universe with 0DTE options, call the `SetFilter` or method. In this example, we will select 0DTE contracts within 1 strike price level and a Delta between 0.25 and 0.75. Then, we will go long on the highest strike price call contract of the SPX index and hold until expiry.

C#

```
public class BasicIndexOptionAlgorithm : QCAlgorithm
{
    private Option _option;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        // Subscribe to the option chain.
        _option = AddIndexOption("SPX", "SPXW");
        // Filter the option universe to only select 0DTE options.
        _option.SetFilter(u => u.IncludeWeeklys().Expiration(0, 0).Strikes(-1, 1));
        // Filter the option universe by Delta. The last SetFilter call prevails.
        // _option.SetFilter(optionFilterUniverse => optionFilterUniverse.Delta(0.25m, 0.75m));
    }

    public override void OnData(Slice slice)
    {
        // Get the option chain data.
        if (Portfolio.Invested || !slice.OptionChains.TryGetValue(_option.Symbol, out var chain))
        {
            return;
        }
        // Sorted the call Option contracts according to their strike prices.
        var call = chain.Where(contract => contract.Right == OptionRight.Call).OrderBy(x =>
x.Strike).FirstOrDefault();
        if (call == null) return;

        // Buy 1 0DTE call option contract for the SPX index.
        Buy(call.Symbol, 1);
    }
}
```

Requesting Data

Individual Contracts

Introduction

The `AddOptionContract` method enables you to add an individual Option contract to your algorithm. To check which contracts are currently available to add to your algorithm, use the `OptionChain` method. If you want to subscribe to a set of contracts instead of individual contracts one-by-one, see Universes .

Create Subscriptions

Before you can subscribe to an Index Option contract, you must configure the underlying Index and get the contract `Symbol` .

```
C#
public class BasicIndexOptionAlgorithm : QCAlgorithm
{
    private Symbol _underlying, _contractSymbol;
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        _underlying = AddIndex("SPX").Symbol;
    }

    public override void OnData(Slice data)
    {
        if (_contractSymbol != null)
        {
            return;
        }
        var chain = OptionChain(_underlying);
        var expiry = chain.Select(contract => contract.Expiry).Min();
        _contractSymbol = chain
            .Where(contract =>
                contract.Expiry == expiry &&
                contract.Right == OptionRight.Call &&
                contract.Greeks.Delta > 0.3m &&
                contract.Greeks.Delta < 0.7m
            )
            .OrderByDescending(contract => contract.OpenInterest)
            .First()
            .Symbol;
        AddOptionContract(_contractSymbol);
    }
}
```

Configure the Underlying Index

In most cases, you should subscribe to the underlying Index before you subscribe to an Index Option contract.

```
C#
_underlying = AddIndex("SPX").Symbol;
```

Get Contract Symbols

To subscribe to an Option contract, you need the contract `Symbol` . The preferred method to getting Option contract `Symbol` objects is to use the `OptionChain` method. This method returns an `OptionChain` object, which represent an entire chain of Option contracts for a single underlying security. With the chain, sort and filter the data to find the specific contract(s) you want to trade.

```
// Get the contracts available to trade.
//   Option A: Standard contracts.
var chain = OptionChain(
    QuantConnect.Symbol.CreateCanonicalOption(_underlying, Market.USA, "?SPX")
);

//   Option B: Weekly contracts.
//var chain = OptionChain(
//    QuantConnect.Symbol.CreateCanonicalOption(_underlying, "SPXW", Market.USA, "?SPXW")
//);

// Select a contract.
var expiry = chain.Select(contract => contract.Expiry).Min();
_contractSymbol = chain
    .Where(contract =>
        // Select call contracts with the closest expiry.
        contract.Expiry == expiry &&
        contract.Right == OptionRight.Call &&
        // Select contracts with a 0.3-0.7 delta.
        contract.Greeks.Delta > 0.3m &&
        contract.Greeks.Delta < 0.7m
    )
    // Select the contract with the largest open interest.
    .OrderByDescending(contract => contract.OpenInterest)
    .First()
    // Get the Symbol of the target contract.
    .Symbol;
```

Alternatively, you can create a Index Option contract `Symbol` object directly parsing its OSI ticker using the `SymbolRepresentation.ParseOptionTickerOSI` method.

```
// Create the contract Symbol for the SPX Call 5650 3 Sept 2024.
_contractSymbol = SymbolRepresentation.ParseOptionTickerOSI("SPXW 240903C05650000", SecurityType.IndexOption,
OptionStyle.European, Market.USA);
```

Subscribe to Contracts

To create an Index Option contract subscription, pass the contract `Symbol` to the `AddIndexOptionContract` method. Save a reference to the contract `Symbol` so you can easily access the contract in the OptionChain that LEAN passes to the `OnData` method. To override the default pricing model of the Option, set a pricing model .

```
var option = AddIndexOptionContract(_contractSymbol);
option.PriceModel = OptionPriceModels.BlackScholes();
```

The `AddIndexOptionContract` method creates a subscription for a single Index Option contract and adds it to your user-defined universe. To create a dynamic universe of Index Option contracts, add an Index Option universe .

Warm Up Contract Prices

If you subscribe to an Index Option contract with `AddIndexOptionContract` , you'll need to wait until the next `Slice` to receive data and trade the contract. To trade the contract in the same time step you subscribe to the contract, set the current price of the contract in a security initializer .

```
Settings.SeedInitialPrices = true;
```

Supported Assets

To view the supported assets in the US Index Options dataset, see Supported Assets .

Resolutions

The following table shows the available resolutions and data formats for Index Option contract subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`				
`Second`				
`Minute`	✓	✓		
`Hour`	✓	✓		
`Daily`	✓	✓		

The default resolution for Index Option subscriptions is `Resolution.Minute` . To change the resolution, pass a `resolution` argument to the `AddIndexOptionContract` method.

```
AddIndexOptionContract(_contractSymbol, Resolution.Hour);
```

To create custom resolution periods, see Consolidating Data .

Supported Markets

The only market available for Index Options is `Market.USA` . You don't need to pass a `Market` argument to the `AddIndexOptionContract` method because the contract `Symbol` already contains the market.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
AddIndexOptionContract(_contractSymbol, fillForward: false);
```

Margin and Leverage

LEAN models buying power and margin calls to ensure your algorithm stays within the margin requirements. Options are already leveraged products, so you can't change their leverage.

Data Normalization

The data normalization mode doesn't affect the data that LEAN passes to `OnData` or the data from history request . If you change the data normalization mode, it won't change the outcome.

Remove Subscriptions

To remove a contract subscription that you created with `AddIndexOptionContract` , call the `RemoveOptionContract` method. This method is an alias for `RemoveSecurity` .

```
RemoveOptionContract(_contractSymbol);
```

The `RemoveOptionContract` method cancels your open orders for the contract and liquidates your holdings.

Properties

The `AddIndexOptionContract` method returns an `Option` object, which have the following properties:

Helper Methods

The `Option` object provides methods you can use for basic calculations. These methods require the underlying price. To get the `Option` object and the `Security` object for its underlying in any function, use the Option `Symbol` to access the value in the `Securities` object.

```
var option = Securities[_contractSymbol];
var underlying = Securities[_contractSymbol.Underlying];
var underlyingPrice = underlying.Price;
```

To get the Option payoff , call the `GetPayOff` method.

```
var payOff = option.GetPayOff(underlyingPrice);
```

To get the Option intrinsic value , call the `GetIntrinsicValue` method.

```
var intrinsicValue = option.GetIntrinsicValue(underlyingPrice);
```

To get the Option out-of-the-money amount , call the `OutOfTheMoneyAmount` method.

```
var otmAmount = option.OutOfTheMoneyAmount(underlyingPrice);
```

To check whether the Option can be automatic exercised, call the `IsAutoExercised` method.

```
var isAutoExercised = option.IsAutoExercised(underlyingPrice);
```

Exceptions and Edge Cases

The following sections explain exceptions and edge cases with subscribing to individual Option contracts.

Manually Creating Option Symbol Objects

To subscribe to an Option contract, you need the contract `Symbol` . To get Index Option contract `Symbol` objects, call the `CreateOption` or `OptionChain` methods. If you use the `CreateOption` method, you need to know the specific contract details.

```
// Standard contracts
_contractSymbol = QuantConnect.Symbol.CreateOption(_underlying, Market.USA,
    OptionStyle.European, OptionRight.Call, 3650, new DateTime(2022, 6, 17));

// Weekly contracts
_weeklyContractSymbol = QuantConnect.Symbol.CreateOption(_underlying, "SPXW", Market.USA,
    OptionStyle.European, OptionRight.Call, 3650, new DateTime(2022, 6, 17));
```

Default Underlying Subscription Settings

If you subscribe to an Index Option contract but don't have a subscription to the underlying Index, LEAN automatically subscribes to the underlying Index and sets its fill forward property to match that of the Index Option contract. In this case, you still need the Index `Symbol` to subscribe to Index Option contracts. If you don't have access to it, create it.

```
_underlying = QuantConnect.Symbol.Create("SPX", SecurityType.Index, Market.USA);
```

Overriding the Initial Implied Volatility Guess

To override the initial guess of implied volatility , set and warm up the underlying volatility model .

Examples

The following examples demonstrate common practices for requesting individual index option contract data.

Example 1: 5-Minute Option Chain

The following example shows how to update the Option chain every five minutes. The custom `OptionChainManager` class implements the selection logic and manages the contract subscriptions.

```
public class OptionChainFullExample : QCAlgorithm
{
    private Dictionary<Symbol, OptionChainManager> _chainManager = new();
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 10, 31);
        SetCash(500000);
        UniverseSettings.Asynchronous = true;
        UniverseSettings.MinimumTimeInUniverse = TimeSpan.Zero;

        // Seed the last price to ensure filtering using the underlying price is available immediately.
        Settings.SeedInitialPrices = true;

        // Request the index underlying data for filtering.
        var spx = AddIndex("SPX").Symbol;
        // Create the 5-minute option chain manager on the SPXW weekly index option.
        _chainManager[QuantConnect.Symbol.CreateCanonicalOption(spx, "SPXW", Market.USA, "?SPXW")] = new(-10, 10, 0,
1);
        // Populate the updated option chain immediately to trade with.
        PopulateOptionChain();

        // Set a schedule event to populate the option chain when the market opens since the option contracts are
updated daily.
        Schedule.On(DateRules.EveryDay(spx), TimeRules.AfterMarketOpen(spx, 1), PopulateOptionChain);
        // Set a scheduled event to filter the closed ATM calls every 5 minutes.
        Schedule.On(DateRules.EveryDay(spx), TimeRules.Every(TimeSpan.FromMinutes(5)), Filter);
    }

    private void PopulateOptionChain()
    {
        // The contract list is updated daily, so we can get it and apply
        // the expiration filter as soon as the market opens.
        foreach (var (symbol, manager) in _chainManager)
        {
            manager.SetChain(OptionChain(symbol), Time);
```

```
        }
        Filter();
    }

    private void Filter()
    {
        foreach (var (symbol, manager) in _chainManager)
        {
            manager.Select(this, symbol);
        }
    }

    public override void OnData(Slice slice)
    {
        foreach (var (symbol, manager) in _chainManager)
        {
            // Only trade on updated data.
            if (!slice.OptionChains.TryGetValue(symbol, out var chain))
                continue;

            // Filter the closest ATM call contract and trade.
            var expiry = chain.Min(x => x.Expiry);
            var atmCall = chain
                .Where(x => x.Expiry == expiry && x.Right == OptionRight.Call && Securities[x.Symbol].IsTradable)
                .OrderBy(x => Math.Abs(chain.Underlying.Price - x.Strike))
                .FirstOrDefault();

            if (atmCall != null && !Portfolio[atmCall.Symbol].Invested)
                MarketOrder(atmCall.Symbol, 1);
        }
    }
}

internal class OptionChainManager
{
    private readonly int _minStrike;
    private readonly int _maxStrike;
    private readonly int _minExpiry;
    private readonly int _maxExpiry;
    private List<OptionContract> _chain = new();
    private readonly List<Symbol> _symbols = new();

    public OptionChainManager(int minStrike, int maxStrike, int minExpiry, int maxExpiry)
    {
        _minStrike = minStrike;
        _maxStrike = maxStrike;
        _minExpiry = minExpiry;
        _maxExpiry = maxExpiry;
    }

    public void SetChain(OptionChain chain, DateTime time)
    {
        // Filter the expiry daily only since the contract list is updated daily.
        _chain = chain.Where(x =>
        {
            var totalDays = (x.Expiry - time).TotalDays;
            return _minExpiry <= totalDays && totalDays <= _maxExpiry;
        }).ToList();
    }

    public void Select(QCAlgorithm algorithm, Symbol underlyingSymbol)
    {
        if (_chain.IsNullOrEmpty())
            return;
        if (underlyingSymbol.IsCanonical())
            underlyingSymbol = underlyingSymbol.Underlying;

        // Filter the contracts with strike range spread between the preset level.
        var strikes = _chain.Select(x => x.Strike).OrderBy(x => x).Distinct().ToList();
        var spot = algorithm.Securities[underlyingSymbol].Price;
        var atm = strikes.OrderBy(x => Math.Abs(spot - x)).FirstOrDefault();
        var index = strikes.IndexOf(atm);
        var minStrike = strikes[Math.Max(0, index + _minStrike)];
        var maxStrike = strikes[Math.Min(strikes.Count - 1, index + _maxStrike)];
        var symbols = _chain
            .Where(x => minStrike <= x.Strike && x.Strike <= maxStrike)
            .Select(x => x.Symbol).ToList();

        // Remove subscriptions if the contracts are no longer ATMs.
        foreach (var symbol in _symbols.Except(symbols).ToList())
        {
```

```
            if (algorithm.RemoveOptionContract(symbol))
                _symbols.Remove(symbol);
        }
        // Request data of the newly identified ATM contracts.
        foreach (var symbol in symbols.Except(_symbols).ToList())
        {
            _symbols.Add(symbol);
            algorithm.AddOptionContract(symbol);
        }
    }
}
```

Index Options

Handling Data

Introduction

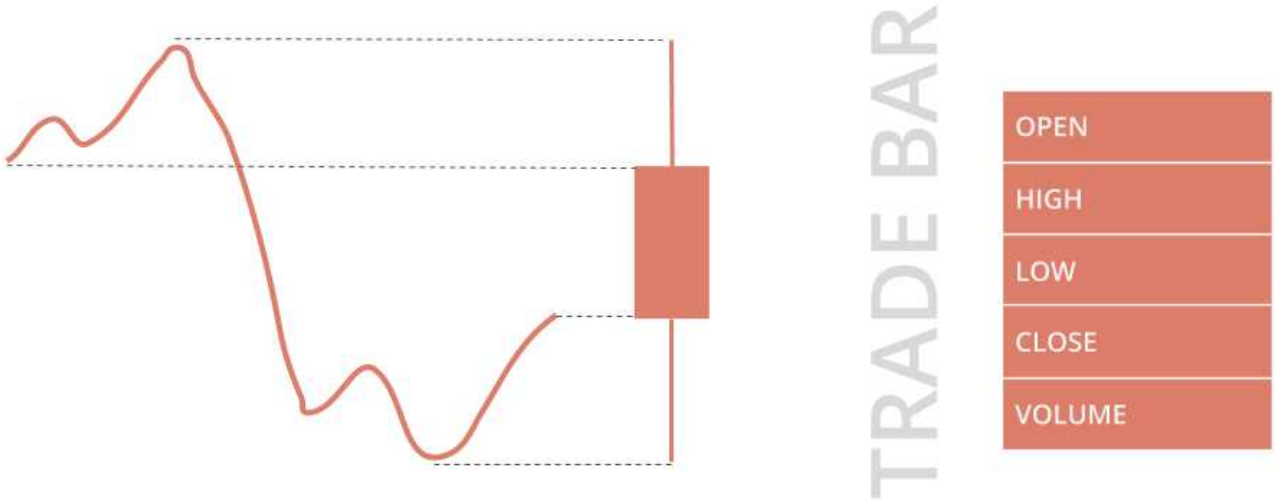
LEAN passes the data you request to the `OnData` method so you can make trading decisions. The default `OnData` method accepts a `Slice` object, but you can define additional `OnData` methods that accept different data types. For example, if you define an `OnData` method that accepts a `TradeBar` argument, it only receives `TradeBar` objects. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `TradeBars` `DataDictionary` is made up of `TradeBar` objects. To access individual data points in the dictionary, you can index the dictionary with the contract ticker or `Symbol` , but we recommend you use the `Symbol` .

To view the resolutions that are available for Index Options data, see Resolutions .

Trades

`TradeBar` objects are price bars that consolidate individual trades from the exchanges. They contain the open, high, low, close, and volume of trading activity over a period of time.



To get the `TradeBar` objects in the `Slice` , index the `Slice` or index the `Bars` property of the `Slice` with the contract `Symbol` . If the contract doesn't actively trade or you are in the same time step as when you added the contract subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your contract before you index the `Slice` with the contract `Symbol` .

```C#
public override void OnData(Slice slice)
{
    // Check if the symbol is contained in TradeBars object
    if (slice.Bars.ContainsKey(_contractSymbol))
    {
        // Obtain the mapped TradeBar of the symbol
        var tradeBar = slice.Bars[_contractSymbol];
    }
}
```

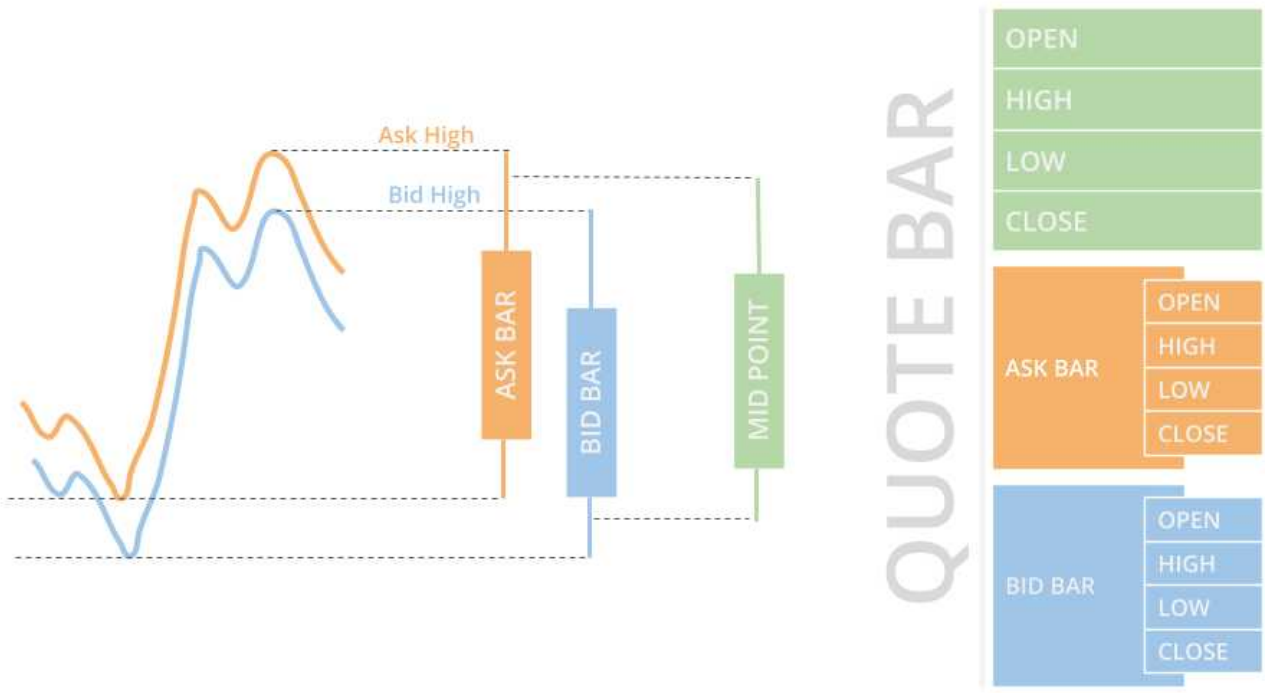
You can also iterate through the `TradeBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `TradeBar` objects.

```C#
public override void OnData(Slice slice)
{
    // Iterate all received Symbol-TradeBar key-value pairs
    foreach (var kvp in slice.Bars)
    {
        var symbol = kvp.Key;
        var tradeBar = kvp.Value;
        var closePrice = tradeBar.Close;
    }
}
```

`TradeBar` objects have the following properties:

Quotes

`QuoteBar` objects are bars that consolidate NBBO quotes from the exchanges. They contain the open, high, low, and close prices of the bid and ask. The `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` object are the mean of the respective bid and ask prices. If the bid or ask portion of the `QuoteBar` has no data, the `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` copy the values of either the `Bid` or `Ask` instead of taking their mean.



To get the `QuoteBar` objects in the `Slice` , index the `QuoteBars` property of the `Slice` with the contract `Symbol` . If the contract doesn't actively get quotes or you are in the same time step as when you added the contract subscription, the `Slice` may not

contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your contract before you index the `Slice` with the contract `Symbol` .

```
C#
public override void OnData(Slice slice)
{
    // Check if the symbol is contained in QuoteBars object
    if (slice.QuoteBars.ContainsKey(_contractSymbol))
    {
        // Obtain the mapped QuoteBar of the symbol
        var quoteBar = slice.QuoteBars[_contractSymbol];
    }
}
```

You can also iterate through the `QuoteBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `QuoteBar` objects.

```
C#
public override void OnData(Slice slice)
{
    // Iterate all received Symbol-QuoteBar key-value pairs
    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        var askPrice = quoteBar.Ask.Close;
    }
}
```

`QuoteBar` objects let LEAN incorporate spread costs into your simulated trade fills to make backtest results more realistic.

`QuoteBar` objects have the following properties:

Option Chains

`OptionChain` objects represent an entire chain of Option contracts for a single underlying security.

To get the `OptionChain` , index the `OptionChains` property of the `Slice` with the canonical `Symbol` .

```
C#
public override void OnData(Slice slice)
{
    // Try to get the OptionChain using the canonical symbol
    if (slice.OptionChains.TryGetValue(_contractSymbol.Canonical, out var chain))
    {
        // Get all contracts if the OptionChain contains any member
        var contracts = chain.Contracts;
    }
}
```

You can also loop through the `OptionChains` property to get each `OptionChain` .

```
C#
public override void OnData(Slice slice)
{
    // Iterate all received Canonical Symbol-OptionChain key-value pairs
    foreach (var kvp in slice.OptionChains)
    {
        var canonicalSymbol = kvp.Key;
        var chain = kvp.Value;
        var contracts = chain.Contracts;
    }
}
```

`OptionChain` objects have the following properties:

Option Contracts

`OptionContract` objects represent the data of a single Option contract in the market. To get the Option contracts in the `Slice` , use the `Contracts` property of the `OptionChain` .

```
C#
public override void OnData(Slice slice)
{
    // Try to get the OptionChain using the canonical symbol
    if (slice.OptionChains.TryGetValue(_contractSymbol.Canonical, out var chain))
    {
        // Get individual contract data
        if (chain.Contracts.TryGetValue(_contractSymbol, out var contract))
        {
            var price = contract.Price;
        }
    }
}
```

Greeks and Implied Volatility

To get the Greeks and implied volatility of an Option contract, use the `Greeks` and `implied_volatility` members.

```
C#
public override void OnData(Slice slice)
{
    // Try to get the OptionChain using the canonical symbol
    if (slice.OptionChains.TryGetValue(_contractSymbol.Canonical, out var chain))
    {
        // Get individual contract data
        if (chain.Contracts.TryGetValue(_contractSymbol, out var contract))
        {
            // Get greeks data of the selected contract
            var delta = contract.Greeks.Delta;
            var iv = contract.ImpliedVolatility;
        }
    }
}
```

LEAN only calculates Greeks and implied volatility when you request them because they are expensive operations. If you invoke the `Greeks` property, the Greeks aren't calculated. However, if you invoke the `Greeks.Delta` , LEAN calculates the delta. To avoid unecessary computation in your algorithm, only request the Greeks and implied volatility when you need them. For more information about the Greeks and implied volatility, see Options Pricing .

Open Interest

Open interest is the number of outstanding contracts that haven't been settled. It provides a measure of investor interest and the market liquidity, so it's a popular metric to use for contract selection. Open interest is calculated once per day. To get the latest open interest value, use the `OpenInterest` property of the `Option` or `OptionContract` .

```C#
public override void OnData(Slice slice)
{
    // Try to get the OptionChains using the canonical symbol
    if (slice.OptionChains.TryGetValue(_contractSymbol.Canonical, out var chain))
    {
        // Get individual contract data
        if (chain.Contracts.TryGetValue(_contractSymbol, out var contract))
        {
            // Get the open interest of the selected contracts
            var openInterest = contract.OpenInterest;
        }
    }
}

public void OnData(OptionChains optionChains)
{
    // Try to get the OptionChains using the canonical symbol
    if (optionChains.TryGetValue(_contractSymbol.Canonical, out var chain))
    {
        // Get individual contract data
        if (chain.Contracts.TryGetValue(_contractSymbol, out var contract))
        {
            // Get the open interest of the selected contracts
            var openInterest = contract.OpenInterest;
        }
    }
}
```

Properties

`OptionContract` objects have the following properties:

Examples

The following examples demonstrate some common practices for handling Index Options.

Example 1: Supply-Demand Pressure Of Individual Contract

This example shows how to handle `QuoteBar` data for the SPXW Index Option contracts to trade based on the supply-demand pressure, while filter individual option contracts and request data using `OptionChain` method for the contracts that expire that day.

C#

```
public class IndexOptionHandlingDataAlgorithm : QCAlgorithm
{
    private Option _spxw;
    private List<Symbol> _contracts = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000000);

        // Subscribe to underlying data for ATM calculation using the updated underlying price.
        // Set data normalization mode to raw is required to ensure the strike price and underlying price are
comparable.
        _spxw = AddIndexOption("SPX", "SPXW");
        _spxw.SetFilter((x) => x.IncludeWeeklys().Expiration(0, 3).Strikes(-5, 5));

        // Update the tradable contracts daily before the market opens since the option contract list provider
populates them daily.
        Schedule.On(
            DateRules.EveryDay(_spxw.Symbol),
            TimeRules.At(9, 0),
            UpdateContracts
        );
    }

    private void UpdateContracts()
    {
        // Get all contracts that expire today, and subscribe to data for trading needs.
        // We do not need to liquidate or handle settlement since all contracts are cash-settled after market close.
        // We only get the strike contracts within $10 of the underlying for higher liquidity and less slippage.
        _contracts = OptionChain(_spxw.Symbol)
            .Where(x => x.Expiry <= Time.AddDays(1) && Math.Abs(x.UnderlyingLastPrice - x.Strike) < 10)
            .Select(x => x.Symbol)
            .ToList();
    }

    public override void OnData(Slice slice)
    {
        // Only focus on a filtered list of option contracts to trade and select the ATM call.
        var contract = _contracts.Where(x => x.ID.OptionRight == OptionRight.Call)
            .MinBy(x => Math.Abs(Securities[x.Underlying].Price - x.ID.StrikePrice));
        if (contract == null)
        {
            return;
        }

        // Trade based on the updated data.
        if (slice.QuoteBars.TryGetValue(contract, out var quote) && !Portfolio.Invested)
        {
            if (quote.Bid != null && quote.Ask != null)
            {
                // If the last trading price is close to the asking price, the selling pressure is higher, so we are
short.
                var diff = quote.Ask.Close - 2 * quote.Close + quote.Bid.Close;
                if (diff < 0)
                {
                    SetHoldings(contract, -0.05m);
                }
                else
                {
                    SetHoldings(contract, 0.05m);
                }
            }
        }
    }
}
```

Example 2: Supply-Demand Pressure Of Universe

This example shows how to handle `QuoteBar` data for shortlisted Index Option contracts to calculate mid-price using bid close and ask close data, while request data through universe selection function using `SetFilter` method for the contracts that expire within the current week. Using mid-price, we can examine the market fair value of the Option and compare it with the model theoretical price.

C#

```
public class IndexOptionHandlingDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 15);
        SetCash(100000000);

        // Subscribe to SPXW option data.
        var option = AddIndexOption("SPX", "SPXW");
        _symbol = option.Symbol;
        // We wish only to trade the contracts expiring within the same week since they have the highest volume.
        option.SetFilter(u => u.IncludeWeeklys().Contracts(x => x.Where(s => s.ID.Date <= Expiry.EndOfWeek(Time))));
    }

    public override void OnData(Slice slice)
    {
        // Only want to obtain the option chain of the selected symbol.
        if (slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            // Select the ATM call to trade.
            var contract = chain.Where(x => x.Right == OptionRight.Call)
                .MinBy(x => Math.Abs(x.UnderlyingLastPrice - x.Strike));

            // If the last trading price is close to the asking price, the selling pressure is higher, so we are
short.
            var diff = contract.AskPrice - 2m * contract.LastPrice + contract.BidPrice;
            if (diff < 0)
            {
                SetHoldings(contract, 0.05m);
            }
            else
            {
                SetHoldings(contract, -0.05m);
            }
        }
    }
}
```

Example 3: Get Instant Delta

The option Greeks change rapidly, so we need to obtain the instant Greeks to calculate the hedge size for arbitration or replication portfolio accurately. You can call the `Greeks` property from the `OptionChain` data to access various Greeks. In this example, we will demonstrate how to obtain a contract with a Delta closest to 0.4 among all call contracts expiring the same week.

C#

```
public class IndexOptionHandlingDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Subscribe to SPXW option data.
        var option = AddIndexOption("SPX", "SPXW");
        _symbol = option.Symbol;
        // We wish only to trade the contracts expiring within the same week since they have the highest volume.
        // 0.4 Delta will only appear in call contracts.
        option.SetFilter(u => u.IncludeWeeklys().Delta(0.2m, 0.6m).Contracts((x) => x.Where(s => s.ID.Date <=
Expiry.EndOfWeek(Time))));
    }

    public override void OnData(Slice slice)
    {
        // Only want to obtain the option chain of the selected symbol.
        if (slice.OptionChains.TryGetValue(_symbol, out var chain) && !Portfolio.Invested)
        {
            // Get the contract with Delta closest to 0.4 to trade.
            // The arbitrary delta criterion might be set due to hedging need or risk adjustment.
            var selected = chain.OrderBy(x => Math.Abs(x.Greeks.Delta - 0.4m)).First();

            // Hold a naked call on the selected contract to speculate an uprise of the SPX index with leverage.
            MarketOrder(selected.Symbol, 1);
        }
    }
}
```

Index Options

Greeks and Implied Volatility

The Greeks are measures that describe the sensitivity of an Option's price to various factors like underlying price changes (Delta), time decay (Theta), volatility (Vega), and interest rates (Rho), while Implied Volatility (IV) represents the market's expectation of the underlying asset's volatility over the life of the Option.

Key Concepts

Concepts on Greeks, IV, and IV modeling

Indicators

Implementation of IV and Greeks as LEAN indicators

See Also

Option Pricing
Risk Free Interest Rate
Dividend Yield

Greeks and Implied Volatility

Key Concepts

Introduction

LEAN provides multiple ways for you to get the Greeks and implied volatility (IV) of Option contracts. There is a daily, pre-calculated dataset based on the end of the previous trading day. There is also a stream of values that are calculated on-the-fly as your algorithm runs.

Greeks on Universes

The Greeks and IV values in the filter function are the daily, pre-calculated values based on the end of the previous trading day.

```C#
public override void Initialize()
{
    var option = AddIndexOption("SPX");
    option.SetFilter(universe => universe.IncludeWeeklys().Delta(0.3m, 0.7m).Expiration(0, 7));
}
```

To calculate the values, we use our implementation of the Black-Scholes pricing model, which accounts for the interest rate and dividend payments when applicable. For example, VIX doesn't have dividends. We use SPY for SPX and QQQ for NDX.

You can't customize the Greeks and IV values that the filter function receives. However, you can create indicators to customize how the Greeks and IV are calculated for the contracts already in your universe.

Greek History

The Greeks and IV values that you get from a history request of the Option universe are the daily, pre-calculated values based on the end of the previous trading day.

```C#
public class OptionHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        var option = AddIndexOption("SPX");
        var history = History<OptionUniverse>(option.Symbol, 5);
        foreach (var chain in history)
        {
            var endTime = chain.EndTime;
            var filteredContracts = chain.Data
                .Select(contract => contract as OptionUniverse)
                .Where(contract => contract.Greeks.Delta > 0.3m);
            foreach (var contract in filteredContracts)
            {
                var price = contract.Price;
                var iv = contract.ImpliedVolatility;
            }
        }
    }
}
```

You can't customize the Greeks and IV values that you get from a history request. However, you can create indicators to customize how the Greeks and IV are calculated for the contracts already in your universe.

Greeks on Data Events

The Greeks and IV values that you get from the current Slice are from the price model calculations. These values aren't the same as the values you get in the universe filter function.

```
C#
public class OptionDataEventsAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        var option = AddIndexOption("VIX", "VIXW");
        option.PriceModel = OptionPriceModels.CrankNicolsonFD();
    }

    public override void OnData(Slice data)
    {
        foreach (var chain in data.OptionChains.Values)
        {
            foreach (var contract in chain)
            {
                var iv = contract.ImpliedVolatility;
                var delta = contract.Greeks.Delta;
            }
        }
    }
}
```

To view all the models LEAN supports, see Supported Models .

Greek Indicators

The Greeks and IV values that you get from the indicators are calculated in the same way as the values you get in the filter function, but they can reflect intraday prices instead of just the daily closing prices.

Examples

For detailed examples on using Option Greeks and Implied Volatility indicators in algorithm, you may refer to the Greeks Indicators page examples.

Greeks and Implied Volatility

Indicators

Introduction

Option indicators enable you to calculate daily and intraday values for the Greeks and implied volatility of Option contracts. This page explains the most popular Option indicators and how to use them with LEAN.

Parameters

The following table describes the arguments that the automatic Option indicators methods accept:

Argument	Data Type	Description	Default Value
`symbol`	`Symbol`	The contract to use when calculating the indicator values.	
`mirrorOption`	`Symbol`	The mirror contract to use in parity type calculations.	`null`
`riskFreeRate`	`decimal`	The risk-free interest rate. If you don't provide a value, the default value is the US primary credit rate from the Interest Rate Provider Model .	`null`
`dividendYield`	`decimal`	The dividend yield rate. If you don't provide a value, the default value comes from the Dividend Yield Provider Model .	`null`
`optionModel`	`OptionPricingModelType`	The Option pricing model that's used to calculate the Greeks. If you don't provide a value, the default value is `OptionPricingModelType.BlackScholes` for Index Options.	`null`
`resolution`	`Resolution`	The resolution of the indicator data. If you don't provide a value, the default value is the resolution of the subscription you have for the Option contract(s).	`null`

To perform implied volatility (IV) smoothing with a put-call pair, pass one of the contracts as the `symbol` argument and pass the other contract in the pair as the `mirrorOption` argument. The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, call the `SetSmoothingFunction` method of the `ImpliedVolatility` class/property.

Several different Option pricing models are supported to calculate the IV and Greeks. The following table describes the

`OptionPricingModelType` enumeration members:

Implied Volatility

Implied volatility, σ , is the market's expectation for the future volatility of an asset and is implied by the price of the assets's Options contracts. You can't observe it in the market but you can derive it from the price of an Option. For more information about implied volatility, see Implied Volatility .

Automatic Indicators

To create an automatic indicator for implied volatility, call the `QCAlgorithm.IV` method with the Option contract `Symbol` object(s).

```
C#
public class AutomaticImpliedVolatilityIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddIndex("SPX").Symbol;

        // Set up a Scheduled Event to select contracts and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count() > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddIndexOptionContract(contracts[0]) as dynamic;
                var contract2 = AddIndexOptionContract(contracts[1]) as dynamic;

                // Create and save the automatic ImpliedVolatility indicators.
                contract1.IV = IV(contracts[0], contracts[1]);
                contract2.IV = IV(contracts[1], contracts[0]);

                _options = (contracts[0], contracts[1]);
            }
        }
    }
```

```
    public override void OnData(Slice slice)
    {
        // Get the ImpliedVolatility indicator of each contract.
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var symbol in chain)
            {
                var option = Securities[symbol] as dynamic;
                var indicator = option.IV;
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

For more information about the `IV` method, see Parameters and Using IV Indicator .

Manual Indicators

To create a manual indicator for implied volatility, call the `ImpliedVolatility` constructor.

C#

```
public class ManualImpliedVolatilityIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private DividendYieldProvider _dividendYieldProvider;
    // Define the Option pricing model.
    private readonly OptionPricingModelType _optionPricingModel = OptionPricingModelType.ForwardTree;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddIndex("SPX").Symbol;

        // Set up dividend yield provider for the underlying
        _dividendYieldProvider = new(_underlying);

        // Set up a Scheduled Event to select contract and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
```

```
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddIndexOptionContract(contracts[0]);
                var contract2 = AddIndexOptionContract(contracts[1]);

                // Create and save the manual ImpliedVolatility indicators.
                foreach (var (contractA, contractB) in new[] { (contract1, contract2), (contract2, contract1) })
                {
                    var option = contractA as dynamic;
                    option.ImpliedVolatility = new ImpliedVolatility(
                            contractA.Symbol, RiskFreeInterestRateModel, _dividendYieldProvider,
                            contractB.Symbol, _optionPricingModel
                        );
                }

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var option in chain)
            {
                var indicator = (Securities[option] as dynamic).ImpliedVolatility;
                var underlying = option.Symbol.Underlying;
                var mirror = QuantConnect.Symbol.CreateOption(
                    underlying.Value, option.ID.Market, option.ID.OptionStyle,
                    option.ID.OptionRight == OptionRight.Call ? OptionRight.Put : OptionRight.Call,
                    option.ID.StrikePrice, option.ID.Date
                ).Value;

                // Check if price data is available for both contracts and the underlying asset.
                var q = slice.QuoteBars;
                var b = slice.Bars;
                if (q.ContainsKey(option) && q.ContainsKey(mirror) && b.ContainsKey(underlying))
                {
                    var dataPoints = new List<IndicatorDataPoint>
                    {
                        new IndicatorDataPoint(option, q[option].EndTime, q[option].Close),
                        new IndicatorDataPoint(mirror, q[mirror].EndTime, q[mirror].Close),
                        new IndicatorDataPoint(
                            underlying, b[underlying].EndTime, b[underlying].Close
                        )
                    };
                    foreach (var dataPoint in dataPoints)
                    {
                        indicator.Update(dataPoint);
                    }

                    // Get the current value of the ImpliedVolatility indicator.
                    var value = indicator.Current.Value;
                }
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
```

```
            Liquidate(_underlying);
        }
    }
}
```

For more information about the `ImpliedVolatility` constructor, see Using IV Indicator .

Volatility Smoothing

The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, pass a `mirrorOption` argument to the `IV` method or `ImpliedVolatility` constructor and then call the `SetSmoothingFunction` method of the resulting `ImpliedVolatility` object. The follow table describes the arguments of the custom function:

Argument	Data Type	Description
`iv`	`decimal`	The IV of the Option contract.
`mirrorIv`	`decimal`	The IV of the mirror Option contract.

The method must return a `decimal` as the smoothened IV.

C#

```
private ImpliedVolatility _iv;

public override void Initialize()
{
    var option = QuantConnect.Symbol.CreateOption("AAPL", Market.USA, OptionStyle.American, OptionRight.Put, 505m,
new DateTime(2014, 6, 27));
    AddOptionContract(option);

    var mirrorOption = QuantConnect.Symbol.CreateOption("AAPL", Market.USA, OptionStyle.American, OptionRight.Call,
505m, new DateTime(2014, 6, 27));
    AddOptionContract(mirrorOption);

    _iv = IV(option, mirrorOption);
    // example: take average of the call-put pair
    _iv.SetSmoothingFunction((iv, mirrorIv) => (iv + mirrorIv) * 0.5m);
}
```

Delta

Delta, Δ , is the rate of change of the Option price with respect to the price of the underlying asset. It measures the first-order sensitivity of the price to a movement in underlying price. For example, an Option delta of 0.4 means that if the underlying asset moves by 1%, then the value of the Option moves by $0.4 \times 1\% = 0.4\%$. For more information about delta, see Delta .

Automatic Indicators

To create an automatic indicator for delta, call the `QCAlgorithm.D` method with the Option contract `Symbol` object(s).

C#

```
public class AutomaticDeltaIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset
```

```
        // Subscribe to the underlying asset.
        _underlying = AddIndex("SPX").Symbol;

        // Set up a Scheduled Event to select contracts and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count() > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddIndexOptionContract(contracts[0]) as dynamic;
                var contract2 = AddIndexOptionContract(contracts[1]) as dynamic;

                // Create and save the automatic Delta indicators.
                contract1.D = D(contracts[0], contracts[1]);
                contract2.D = D(contracts[1], contracts[0]);

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        // Get the Delta indicator of each contract.
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var symbol in chain)
            {
                var option = Securities[symbol] as dynamic;
                var indicator = option.D;
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

The follow table describes the arguments that the `D` method accepts in addition to the standard parameters :

Argument	Data Type	Description	Default Value
`ivModel`	`OptionPricingModelType`	The Option pricing model to use to estimate the IV when calculating Delta. If you don't provide a value, the default value is to match the `optionModel` parameter.	`null`

For more information about the `D` method, see Using D Indicator .

Manual Indicators

To create a manual indicator for delta, call the `Delta` constructor.

```
C#
public class ManualDeltaIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private DividendYieldProvider _dividendYieldProvider;
    // Define the Option pricing model.
    private readonly OptionPricingModelType _optionPricingModel = OptionPricingModelType.ForwardTree;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddIndex("SPX").Symbol;

        // Set up dividend yield provider for the underlying
        _dividendYieldProvider = new(_underlying);

        // Set up a Scheduled Event to select contract and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count > 1)
            {
                // Subscribe to both contracts.
```

```
                var contract1 = AddIndexOptionContract(contracts[0]);
                var contract2 = AddIndexOptionContract(contracts[1]);

                // Create and save the manual Delta indicators.
                foreach (var (contractA, contractB) in new[] { (contract1, contract2), (contract2, contract1) })
                {
                    var option = contractA as dynamic;
                    option.Delta = new Delta(
                            contractA.Symbol, RiskFreeInterestRateModel, _dividendYieldProvider,
                            contractB.Symbol, _optionPricingModel
                        );
                }

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var option in chain)
            {
                var indicator = (Securities[option] as dynamic).Delta;
                var underlying = option.Symbol.Underlying;
                var mirror = QuantConnect.Symbol.CreateOption(
                    underlying.Value, option.ID.Market, option.ID.OptionStyle,
                    option.ID.OptionRight == OptionRight.Call ? OptionRight.Put : OptionRight.Call,
                    option.ID.StrikePrice, option.ID.Date
                ).Value;

                // Check if price data is available for both contracts and the underlying asset.
                var q = slice.QuoteBars;
                var b = slice.Bars;
                if (q.ContainsKey(option) && q.ContainsKey(mirror) && b.ContainsKey(underlying))
                {
                    var dataPoints = new List<IndicatorDataPoint>
                    {
                        new IndicatorDataPoint(option, q[option].EndTime, q[option].Close),
                        new IndicatorDataPoint(mirror, q[mirror].EndTime, q[mirror].Close),
                        new IndicatorDataPoint(
                            underlying, b[underlying].EndTime, b[underlying].Close
                        )
                    };
                    foreach (var dataPoint in dataPoints)
                    {
                        indicator.Update(dataPoint);
                    }

                    // Get the current value of the Delta indicator.
                    var value = indicator.Current.Value;
                }
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

For more information about the `Delta` constructor, see Using D Indicator .

Volatility Smoothing

The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, pass a `mirrorOption` argument to the indicator method or constructor and then call the

SetSmoothingFunction method of the ImpliedVolatility property of the indicator.

```
// Example: Average IV of the call-put pair.
deltaIndicator.ImpliedVolatility.SetSmoothingFunction((iv, mirrorIv) => (iv + mirrorIv) * 0.5m);
```

Gamma

Gamma, Γ, is the rate of change of the portfolio's delta with respect to the underlying asset's price. It represents the second-order sensitivity of the Option to a movement in the underlying asset's price. For more information about Gamma, see Gamma .

Automatic Indicators

To create an automatic indicator for gamma, call the QCAlgorithm.G method with the Option contract Symbol object(s).

```
public class AutomaticGammaIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddIndex("SPX").Symbol;

        // Set up a Scheduled Event to select contracts and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count() > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddIndexOptionContract(contracts[0]) as dynamic;
                var contract2 = AddIndexOptionContract(contracts[1]) as dynamic;

                // Create and save the automatic Gamma indicators.
                contract1.G = G(contracts[0], contracts[1]);
                contract2.G = G(contracts[1], contracts[0]);
```

```
                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        // Get the Gamma indicator of each contract.
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var symbol in chain)
            {
                var option = Securities[symbol] as dynamic;
                var indicator = option.G;
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

The follow table describes the arguments that the `G` method accepts in addition to the standard parameters :

Argument	Data Type	Description	Default Value
`ivModel`	`OptionPricingModelType`	The Option pricing model to use to estimate the IV when calculating Gamma. If you don't provide a value, the default value is to match the `optionModel` parameter.	`null`

For more information about the `G` method, see Using G Indicator .

Manual Indicators

To create a manual indicator for gamma, call the `Gamma` constructor.

C#

```
public class ManualGammaIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private DividendYieldProvider _dividendYieldProvider;
    // Define the Option pricing model.
    private readonly OptionPricingModelType _optionPricingModel = OptionPricingModelType.ForwardTree;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddIndex("SPX").Symbol;

        // Set up dividend yield provider for the underlying
        _dividendYieldProvider = new(_underlying);

        // Set up a Scheduled Event to select contract and create the indicators every day before market open
```

```
        // Set up a scheduled event to select contract and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddIndexOptionContract(contracts[0]);
                var contract2 = AddIndexOptionContract(contracts[1]);

                // Create and save the manual Gamma indicators.
                foreach (var (contractA, contractB) in new[] { (contract1, contract2), (contract2, contract1) })
                {
                    var option = contractA as dynamic;
                    option.Gamma = new Gamma(
                            contractA.Symbol, RiskFreeInterestRateModel, _dividendYieldProvider,
                            contractB.Symbol, _optionPricingModel
                        );
                }

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var option in chain)
            {
                var indicator = (Securities[option] as dynamic).Gamma;
                var underlying = option.Symbol.Underlying;
                var mirror = QuantConnect.Symbol.CreateOption(
                    underlying.Value, option.ID.Market, option.ID.OptionStyle,
                    option.ID.OptionRight == OptionRight.Call ? OptionRight.Put : OptionRight.Call,
                    option.ID.StrikePrice, option.ID.Date
                ).Value;

                // Check if price data is available for both contracts and the underlying asset.
                var q = slice.QuoteBars;
                var b = slice.Bars;
                if (q.ContainsKey(option) && q.ContainsKey(mirror) && b.ContainsKey(underlying))
                {
                    var dataPoints = new List<IndicatorDataPoint>
                    {
                        new IndicatorDataPoint(option, q[option].EndTime, q[option].Close),
                        new IndicatorDataPoint(mirror, q[mirror].EndTime, q[mirror].Close),
                        new IndicatorDataPoint(
                            underlying, b[underlying].EndTime, b[underlying].Close
                        )
                    };
```

```
                    };
                    foreach (var dataPoint in dataPoints)
                    {
                        indicator.Update(dataPoint);
                    }

                    // Get the current value of the Gamma indicator.
                    var value = indicator.Current.Value;
                }
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

For more information about the `Gamma` constructor, see Using G Indicator .

Volatility Smoothing

The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, pass a `mirrorOption` argument to the indicator method or constructor and then call the `SetSmoothingFunction` method of the `ImpliedVolatility` property of the indicator.

C#

```
// Example: Average IV of the call-put pair.
gammaIndicator.ImpliedVolatility.SetSmoothingFunction((iv, mirrorIv) => (iv + mirrorIv) * 0.5m);
```

Vega

Vega, ν , is the rate of change in the value of the Option with respect to the volatility of the underlying asset. For more information about vega, see Vega .

Automatic Indicators

To create an automatic indicator for vega, call the `QCAlgorithm.V` method with the Option contract `Symbol` object(s).

C#

```
public class AutomaticVegaIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddIndex("SPX").Symbol;

        // Set up a Scheduled Event to select contracts and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }
```

```
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count() > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddIndexOptionContract(contracts[0]) as dynamic;
                var contract2 = AddIndexOptionContract(contracts[1]) as dynamic;

                // Create and save the automatic Vega indicators.
                contract1.V = V(contracts[0], contracts[1]);
                contract2.V = V(contracts[1], contracts[0]);

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        // Get the Vega indicator of each contract.
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var symbol in chain)
            {
                var option = Securities[symbol] as dynamic;
                var indicator = option.V;
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

The follow table describes the arguments that the `V` method accepts in addition to the standard parameters :

Argument	Data Type	Description	Default Value
ivModel	OptionPricingModelType	The Option pricing model to use to estimate the IV when calculating Vega. If you don't provide a value, the default value is to match the optionModel parameter.	null

For more information about the V method, see Using V Indicator .

Manual Indicators

To create a manual indicator for vega, call the Vega constructor.

```C#
public class ManualVegaIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private DividendYieldProvider _dividendYieldProvider;
    // Define the Option pricing model.
    private readonly OptionPricingModelType _optionPricingModel = OptionPricingModelType.ForwardTree;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddIndex("SPX").Symbol;

        // Set up dividend yield provider for the underlying
        _dividendYieldProvider = new(_underlying);

        // Set up a Scheduled Event to select contract and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count > 1)
            {
                // Subscribe to both contracts.
```

```
                var contract1 = AddIndexOptionContract(contracts[0]);
                var contract2 = AddIndexOptionContract(contracts[1]);

                // Create and save the manual Vega indicators.
                foreach (var (contractA, contractB) in new[] { (contract1, contract2), (contract2, contract1) })
                {
                    var option = contractA as dynamic;
                    option.Vega = new Vega(
                            contractA.Symbol, RiskFreeInterestRateModel, _dividendYieldProvider,
                            contractB.Symbol, _optionPricingModel
                        );
                }

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var option in chain)
            {
                var indicator = (Securities[option] as dynamic).Vega;
                var underlying = option.Symbol.Underlying;
                var mirror = QuantConnect.Symbol.CreateOption(
                    underlying.Value, option.ID.Market, option.ID.OptionStyle,
                    option.ID.OptionRight == OptionRight.Call ? OptionRight.Put : OptionRight.Call,
                    option.ID.StrikePrice, option.ID.Date
                ).Value;

                // Check if price data is available for both contracts and the underlying asset.
                var q = slice.QuoteBars;
                var b = slice.Bars;
                if (q.ContainsKey(option) && q.ContainsKey(mirror) && b.ContainsKey(underlying))
                {
                    var dataPoints = new List<IndicatorDataPoint>
                    {
                        new IndicatorDataPoint(option, q[option].EndTime, q[option].Close),
                        new IndicatorDataPoint(mirror, q[mirror].EndTime, q[mirror].Close),
                        new IndicatorDataPoint(
                            underlying, b[underlying].EndTime, b[underlying].Close
                        )
                    };
                    foreach (var dataPoint in dataPoints)
                    {
                        indicator.Update(dataPoint);
                    }

                    // Get the current value of the Vega indicator.
                    var value = indicator.Current.Value;
                }
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

For more information about the `Vega` constructor, see Using V Indicator .

Volatility Smoothing

The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, pass a `mirrorOption` argument to the indicator method or constructor and then call the

`SetSmoothingFunction` method of the `ImpliedVolatility` property of the indicator.

```
// Example: Average IV of the call-put pair.
vegaIndicator.ImpliedVolatility.SetSmoothingFunction((iv, mirrorIv) => (iv + mirrorIv) * 0.5m);
```

Theta

Theta, Θ , is the rate of change of the value of the Option with respect to the passage of time. It is also known to as the time decay of an Option. For more information about theta, see Theta .

Automatic Indicators

To create an automatic indicator for theta, call the `QCAlgorithm.T` method with the Option contract `Symbol` object(s).

```
public class AutomaticThetaIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddIndex("SPX").Symbol;

        // Set up a Scheduled Event to select contracts and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count() > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddIndexOptionContract(contracts[0]) as dynamic;
                var contract2 = AddIndexOptionContract(contracts[1]) as dynamic;

                // Create and save the automatic Theta indicators.
                contract1.T = T(contracts[0], contracts[1]);
                contract2.T = T(contracts[1], contracts[0]);
```

```
                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        // Get the Theta indicator of each contract.
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var symbol in chain)
            {
                var option = Securities[symbol] as dynamic;
                var indicator = option.T;
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

The follow table describes the arguments that the `T` method accepts in addition to the standard parameters :

Argument	Data Type	Description	Default Value
`ivModel`	`OptionPricingModelType`	The Option pricing model to use to estimate the IV when calculating theta. If you don't provide a value, the default value is to match the `optionModel` parameter.	`null`

For more information about the `T` method, see Using T Indicator .

Manual Indicators

To create a manual indicator for theta, call the `Theta` constructor.

C#

```
public class ManualThetaIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private DividendYieldProvider _dividendYieldProvider;
    // Define the Option pricing model.
    private readonly OptionPricingModelType _optionPricingModel = OptionPricingModelType.ForwardTree;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddIndex("SPX").Symbol;

        // Set up dividend yield provider for the underlying
        _dividendYieldProvider = new(_underlying);

        // Set up a Scheduled Event to select contract and create the indicators every day before market open
```

```
        // Set up a scheduled event to select contract and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddIndexOptionContract(contracts[0]);
                var contract2 = AddIndexOptionContract(contracts[1]);

                // Create and save the manual Theta indicators.
                foreach (var (contractA, contractB) in new[] { (contract1, contract2), (contract2, contract1) })
                {
                    var option = contractA as dynamic;
                    option.Theta = new Theta(
                            contractA.Symbol, RiskFreeInterestRateModel, _dividendYieldProvider,
                            contractB.Symbol, _optionPricingModel
                        );
                }

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var option in chain)
            {
                var indicator = (Securities[option] as dynamic).Theta;
                var underlying = option.Symbol.Underlying;
                var mirror = QuantConnect.Symbol.CreateOption(
                    underlying.Value, option.ID.Market, option.ID.OptionStyle,
                    option.ID.OptionRight == OptionRight.Call ? OptionRight.Put : OptionRight.Call,
                    option.ID.StrikePrice, option.ID.Date
                ).Value;

                // Check if price data is available for both contracts and the underlying asset.
                var q = slice.QuoteBars;
                var b = slice.Bars;
                if (q.ContainsKey(option) && q.ContainsKey(mirror) && b.ContainsKey(underlying))
                {
                    var dataPoints = new List<IndicatorDataPoint>
                    {
                        new IndicatorDataPoint(option, q[option].EndTime, q[option].Close),
                        new IndicatorDataPoint(mirror, q[mirror].EndTime, q[mirror].Close),
                        new IndicatorDataPoint(
                            underlying, b[underlying].EndTime, b[underlying].Close
                        )
                    };
```

```
                    };
                    foreach (var dataPoint in dataPoints)
                    {
                        indicator.Update(dataPoint);
                    }

                    // Get the current value of the Theta indicator.
                    var value = indicator.Current.Value;
                }
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

For more information about the `Theta` constructor, see Using T Indicator .

Volatility Smoothing

The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, pass a `mirrorOption` argument to the indicator method or constructor and then call the `SetSmoothingFunction` method of the `ImpliedVolatility` property of the indicator.

C#

```
// Example: Average IV of the call-put pair.
thetaIndicator.ImpliedVolatility.SetSmoothingFunction((iv, mirrorIv) => (iv + mirrorIv) * 0.5m);
```

Rho

Rho, ρ , is the rate of change of the value of a derivative with respect to the interest rate . It is usually small and not a big issue in practice unless the Option is deep in-the-money and has a long horizon. In this case, the interest rate matters because you need to discount a larger cash flow over a longer horizon. For more information about rho, see Rho .

Automatic Indicators

To create an automatic indicator for rho, call the `QCAlgorithm.R` method with the Option contract `Symbol` object(s).

C#

```
public class AutomaticRhoIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddIndex("SPX").Symbol;

        // Set up a Scheduled Event to select contracts and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
```

```
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count() > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddIndexOptionContract(contracts[0]) as dynamic;
                var contract2 = AddIndexOptionContract(contracts[1]) as dynamic;

                // Create and save the automatic Rho indicators.
                contract1.R = R(contracts[0], contracts[1]);
                contract2.R = R(contracts[1], contracts[0]);

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        // Get the Rho indicator of each contract.
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var symbol in chain)
            {
                var option = Securities[symbol] as dynamic;
                var indicator = option.R;
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

The follow table describes the arguments that the R method accepts in addition to the standard parameters :

Argument	Data Type	Description	Default Value
ivModel	OptionPricingModelType	The Option pricing model to use to estimate the IV when calculating rho If you don't provide a value, the default value is to match the optionModel parameter.	null

For more information about the R method, see Using R Indicator .

Manual Indicators

To create a manual indicator for rho, call the Rho constructor.

C#

```
public class ManualRhoIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _underlying;
    private DividendYieldProvider _dividendYieldProvider;
    // Define the Option pricing model.
    private readonly OptionPricingModelType _optionPricingModel = OptionPricingModelType.ForwardTree;
    private (Symbol option1, Symbol option2) _options;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(500000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Subscribe to the underlying asset.
        _underlying = AddIndex("SPX").Symbol;

        // Set up dividend yield provider for the underlying
        _dividendYieldProvider = new(_underlying);

        // Set up a Scheduled Event to select contract and create the indicators every day before market open.
        Schedule.On(
            DateRules.EveryDay(_underlying),
            TimeRules.At(9, 0),
            UpdateContractsAndGreeks
        );
    }

    private void UpdateContractsAndGreeks()
    {
        if (_underlying == null)
        {
            return;
        }
        // Get all the tradable Option contracts.
        var chain = OptionChain(_underlying);

        // Filter the contracts down. For example, ATM contracts with atleast 1 month until expiry.
        var filteredChain = chain.Where(contract => contract.Expiry > Time.AddDays(30));
        if (filteredChain.Count() == 0)
        {
            return;
        }
        var expiry = filteredChain.Min(contract => contract.Expiry);
        filteredChain = filteredChain.Where(contract => contract.Expiry == expiry);

        var minStrikeDelta = filteredChain.Min(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice));
        filteredChain = filteredChain.Where(contract => Math.Abs(contract.Strike - contract.UnderlyingLastPrice) ==
minStrikeDelta);

        // Group the contracts into call/put pairs.
        foreach (var group in filteredChain.GroupBy(contract => contract.Strike))
        {
            var contracts = group.ToList();
            if (contracts.Count > 1)
            {
                // Subscribe to both contracts.
                var contract1 = AddIndexOptionContract(contracts[0]);
```

```
                var contract2 = AddIndexOptionContract(contracts[1]);

                // Create and save the manual Rho indicators.
                foreach (var (contractA, contractB) in new[] { (contract1, contract2), (contract2, contract1) })
                {
                    var option = contractA as dynamic;
                    option.Rho = new Rho(
                            contractA.Symbol, RiskFreeInterestRateModel, _dividendYieldProvider,
                            contractB.Symbol, _optionPricingModel
                        );
                }

                _options = (contracts[0], contracts[1]);
            }
        }
    }

    public override void OnData(Slice slice)
    {
        foreach (var (canonical, chain) in slice.OptionChains)
        {
            foreach (var option in chain)
            {
                var indicator = (Securities[option] as dynamic).Rho;
                var underlying = option.Symbol.Underlying;
                var mirror = QuantConnect.Symbol.CreateOption(
                    underlying.Value, option.ID.Market, option.ID.OptionStyle,
                    option.ID.OptionRight == OptionRight.Call ? OptionRight.Put : OptionRight.Call,
                    option.ID.StrikePrice, option.ID.Date
                ).Value;

                // Check if price data is available for both contracts and the underlying asset.
                var q = slice.QuoteBars;
                var b = slice.Bars;
                if (q.ContainsKey(option) && q.ContainsKey(mirror) && b.ContainsKey(underlying))
                {
                    var dataPoints = new List<IndicatorDataPoint>
                    {
                        new IndicatorDataPoint(option, q[option].EndTime, q[option].Close),
                        new IndicatorDataPoint(mirror, q[mirror].EndTime, q[mirror].Close),
                        new IndicatorDataPoint(
                            underlying, b[underlying].EndTime, b[underlying].Close
                        )
                    };
                    foreach (var dataPoint in dataPoints)
                    {
                        indicator.Update(dataPoint);
                    }

                    // Get the current value of the Rho indicator.
                    var value = indicator.Current.Value;
                }
            }
        }

        // Sell straddle as an example to trade.
        if (!Portfolio.Invested && _options != default)
        {
            Sell(_options.option1, 1);
            Sell(_options.option2, 1);
        }
        // Liquidate any assigned positions.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
    }
}
```

For more information about the `Rho` constructor, see Using R Indicator .

Volatility Smoothing

The default IV smoothing method uses the one contract in the pair that's at-the-money or out-of-money to calculate the IV. To change the smoothing function, pass a `mirrorOption` argument to the indicator method or constructor and then call the

`SetSmoothingFunction` method of the `ImpliedVolatility` property of the indicator.

```C#
// Example: Average IV of the call-put pair.
rhoIndicator.ImpliedVolatility.SetSmoothingFunction((iv, mirrorIv) => (iv + mirrorIv) * 0.5m);
```

Examples

Demonstration Algorithms
OptionIndicatorsRegressionAlgorithm.cs C# OptionIndicatorsMirrorContractsRegressionAlgorithm.cs C#

Index Options

Market Hours

USA

Market Hours

USA

Introduction

This page shows the trading hours, holidays, and time zone of the US Index Option market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US Index Option market:

Weekday	Time (America/Chicago)
Monday	08:30:00 to 15:20:00
Tuesday	08:30:00 to 15:20:00
Wednesday	08:30:00 to 15:20:00
Thursday	08:30:00 to 15:20:00
Friday	08:30:00 to 15:20:00

Post-market Hours

Post-market trading is not available.

Holidays

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The US Index Option market trades in the following time zones:

- `America/Chicago`
- `America/New York`

Assets With Other Hours

The following table shows the contracts that have different trading periods than the overall US Index Option market:

Symbol	Name
NDX	Nasdaq 100 Index
RUT	Russell 2000 Index
SPX	S&P 500 Index
VIX	CBOE Volatility Index

USA

NDX

Introduction

This page shows the trading hours, holidays, and time zone of the Nasdaq 100 Index Option contracts market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Nasdaq 100 Index Option contracts market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:15:00
Tuesday	09:30:00 to 16:15:00
Wednesday	09:30:00 to 16:15:00
Thursday	09:30:00 to 16:15:00
Friday	09:30:00 to 16:15:00

Post-market Hours

Post-market trading is not available.

Holidays

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Nasdaq 100 Index Option contracts market trades in the `America/New York` time zone.

USA

RUT

Introduction

This page shows the trading hours, holidays, and time zone of the Russell 2000 Index Option contracts market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Russell 2000 Index Option contracts market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:15:00
Tuesday	09:30:00 to 16:15:00
Wednesday	09:30:00 to 16:15:00
Thursday	09:30:00 to 16:15:00
Friday	09:30:00 to 16:15:00

Post-market Hours

Post-market trading is not available.

Holidays

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Russell 2000 Index Option contracts market trades in the `America/New York` time zone.

USA

SPX

Introduction

This page shows the trading hours, holidays, and time zone of the S&P 500 Index Option contracts market.

Pre-market Hours

The following table shows the pre-market hours for the S&P 500 Index Option contracts market:

Weekday	**Time (America/Chicago)**
Sunday	19:15:00 to 24:00:00
Monday	00:00:00 to 08:15:00
Tuesday	00:00:00 to 08:15:00
Wednesday	00:00:00 to 08:15:00
Thursday	00:00:00 to 08:15:00
Friday	00:00:00 to 08:15:00

Regular Trading Hours

The following table shows the regular trading hours for the S&P 500 Index Option contracts market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:15:00
Tuesday	08:30:00 to 15:15:00
Wednesday	08:30:00 to 15:15:00
Thursday	08:30:00 to 15:15:00
Friday	08:30:00 to 15:15:00

Post-market Hours

The following table shows the post-market hours for the S&P 500 Index Option contracts market:

Weekday	Time (America/Chicago)
Monday	15:15:00 to 16:00:00, 19:15:00 to 24:00:00
Tuesday	15:15:00 to 16:00:00, 19:15:00 to 24:00:00
Wednesday	15:15:00 to 16:00:00, 19:15:00 to 24:00:00
Thursday	15:15:00 to 16:00:00, 19:15:00 to 24:00:00
Friday	15:15:00 to 16:00:00

Holidays

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The S&P 500 Index Option contracts market trades in the `America/Chicago` time zone.

USA

VIX

Introduction

This page shows the trading hours, holidays, and time zone of the CBOE Volatility Index Option contracts market.

Pre-market Hours

The following table shows the pre-market hours for the CBOE Volatility Index Option contracts market:

Weekday	**Time (America/Chicago)**
Sunday	19:15:00 to 24:00:00
Monday	00:00:00 to 08:15:00
Tuesday	00:00:00 to 08:15:00
Wednesday	00:00:00 to 08:15:00
Thursday	00:00:00 to 08:15:00
Friday	00:00:00 to 08:15:00

Regular Trading Hours

The following table shows the regular trading hours for the CBOE Volatility Index Option contracts market:

Weekday	**Time (America/Chicago)**
Monday	08:30:00 to 15:15:00
Tuesday	08:30:00 to 15:15:00
Wednesday	08:30:00 to 15:15:00
Thursday	08:30:00 to 15:15:00
Friday	08:30:00 to 15:15:00

Post-market Hours

The following table shows the post-market hours for the CBOE Volatility Index Option contracts market:

Weekday	Time (America/Chicago)
Monday	15:15:00 to 16:00:00, 19:15:00 to 24:00:00
Tuesday	15:15:00 to 16:00:00, 19:15:00 to 24:00:00
Wednesday	15:15:00 to 16:00:00, 19:15:00 to 24:00:00
Thursday	15:15:00 to 16:00:00, 19:15:00 to 24:00:00
Friday	15:15:00 to 16:00:00

Holidays

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CBOE Volatility Index Option contracts market trades in the `America/Chicago` time zone.

Asset Classes

CFD

A contract for difference (CFD) is a contract between a buyer and a seller that stipulates that the buyer must pay the seller the difference between the current value of an asset and its value at contract time.

Requesting Data

Handling Data

Market Hours

See Also

BasicTemplateCfdAlgorithm.py
BasicTemplateCfdAlgorithm.cs

CFD

Requesting Data

Introduction

Request Contract for Difference (CFD) data in your algorithm to receive a feed of contract prices in the `OnData` method. QuantConnect supports CFD trading with OANDA and Interactive Brokers .

OANDA Subscriptions

To create a CFD subscription, in the `Initialize` method, call the `AddCfd` method. The `AddCfd` method returns a `Cfd` Security object, which contains a `Symbol` property. Save a reference to the `Symbol` so you can use it in `OnData` to access the security data in the `Slice` .

```
C#
_symbol = AddCfd("XAUUSD").Symbol;
```

To view the supported CFD contracts, see Supported Assets . For more information about the specific dataset we use for backtests, see the OANDA CFD dataset listing . To trade OANDA CFDs live, use our QuantConnect data provider .

Interactive Brokers Subscriptions

To create a CFD subscription for Interactive Brokers (IB) , in the `Initialize` method, call the `AddCfd` method and set the `market` parameter to `Market.InteractiveBrokers` . The `AddCfd` method returns a `Cfd` Security object, which contains a `Symbol` property. Save a reference to the `Symbol` so you can use it in `OnData` to access the security data in the `Slice` .

```
C#
_symbol = AddCfd("SPY", market: Market.InteractiveBrokers).Symbol;
```

Historical data for backtesting IB CFDs is unavailable.

In live trading, include the Interactive Brokers data provider when you deploy the algorithm to access IB CFD data. You can use this data provider for paper or real-money accounts. IB provides data for Indexes, Metals, Forex, and Global Stock CFDs. To view the Index and Forex CFD products that IB supports in live trading, see the Interactive Brokers Supported Contracts . To view the Equity CFD products, see the Products Search page on the IB website.

If you live in the EU and can't trade US ETFs due to regulation, you can trade their CFD equivalents in live mode.

```
C#
var securityType = LiveMode ? SecurityType.Cfd : SecurityType.Equity;
var market = LiveMode ? Market.InteractiveBrokers : Market.USA;
_symbol = QuantConnect.Symbol.Create("SPY", securityType, Market.InteractiveBrokers);
AddSecurity(_symbol);
```

Market hours for IB CFDs are not modeled. You cannot use them for scheduled events or to get market open/close times . To create a scheduled event, use a ticker of any US Equity that you haven't added as a IB CFD or create a US Equity `Symbol` .

```C#
public override void Initialize()
{
    var usEquitySymbol = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
    // Call the BeforeMarketOpen method 10 minutes before open every day US Equity stocks are trading.
    Schedule.On(
        DateRules.EveryDay(usEquitySymbol),
        TimeRules.BeforeMarketOpen(usEquitySymbol, 10),
        BeforeMarketOpen
    );
}
```

Resolutions

The following table shows the available resolutions and data formats for CFD subscriptions:

Resolution	TradeBar	QuoteBar	Trade Tick	Quote Tick
`Tick`				✓
`Second`		✓		
`Minute`		✓		
`Hour`		✓		
`Daily`		✓		

The default resolution for CFD subscriptions is `Resolution.Minute` . To change the resolution, pass a `resolution` argument to the `AddCfd` method.

```C#
_symbol = AddCfd("XAUUSD", Resolution.Daily).Symbol;
```

To create custom resolution periods, see Consolidating Data .

Supported Markets

The following `Market` enumeration members are available for CFDs:

To set the market for a security, pass a `market` argument to the `AddCfd` method.

```C#
_symbol = AddCfd("XAUUSD", market: Market.Oanda).Symbol;
```

The brokerage models have a default market for each asset class. If you set a brokerage model, you may not need to specify the market to use. In backtesting, set the market for CFDs to OANDA.

Fill Forward

Fill forward means if there is no data point for the current slice , LEAN uses the previous data point. Fill forward is the default data setting. If you disable fill forward, you may get stale fills or you may see trade volume as zero.

To disable fill forward for a security, set the `fillForward` argument to false when you create the security subscription.

```
_symbol = AddCfd("XAUUSD", fillForward: false).Symbol;
```

Margin and Leverage

LEAN models buying power and margin calls to ensure your algorithm stays within the margin requirements. The specific margin available depends on your brokerage destination. To change the amount of leverage you can use for a CFD contract, pass a `leverage` argument to the `AddCfd` method.

```
_symbol = AddCfd("XAUUSD", leverage: 35).Symbol;
```

Data Normalization

The data normalization mode doesn't affect the data that LEAN passes to `OnData` or the data from history request . If you change the data normalization mode, it won't change the outcome.

Interactive Brokers Stock CFDs apply corporate actions to the price of the asset, including paying dividends in cash to your account. This behavior is not modeled today and splits in live trading will result in price discontinuities. You should monitor the underlying Equity for events and use the underlying Equity for indicators.

Properties

The `AddCfd` method returns a `Cfd` object, which have the following properties:

Interactive Brokers Supported Contracts

The following table shows the CFD contracts that IB supports:

Contract Available (107)					
AUDCAD	AUDCHF	AUDCNH	AUDHKD	AUDJPY	AUDNZD
AUDSGD	AUDUSD	AUDZAR	CADCHF	CADCNH	CADHKD
CADJPY	CHFCNH	CHFCZK	CHFDKK	CHFHUF	CHFJPY
CHFNOK	CHFPLN	CHFSEK	CHFZAR	CNHHKD	CNHJPY
DKKJPY	DKKNOK	DKKSEK	EURAUD	EURCAD	EURCHF
EURCNH	EURCZK	EURDKK	EURGBP	EURHKD	EURHUF
EURILS	EURJPY	EURMXN	EURNOK	EURNZD	EURPLN
EURRUB	EURSEK	EURSGD	EURUSD	EURZAR	GBPAUD
GBPCAD	GBPCHF	GBPCNH	GBPCZK	GBPDKK	GBPHKD
GBPHUF	GBPJPY	GBPMXN	GBPNOK	GBPNZD	GBPPLN
GBPSEK	GBPSGD	GBPUSD	GBPZAR	HKDJPY	IBAU200
IBCH20	IBDE40	IBES35	IBEU50	IBFR40	IBGB100
IBHK50	IBJP225	IBNL25	IBUS30	IBUS500	IBUST100
MXNJPY	NOKJPY	NOKSEK	NZDCAD	NZDCHF	NZDJPY
NZDUSD	SEKJPY	SGDCNH	SGDJPY	USDCAD	USDCHF
USDCNH	USDCZK	USDDKK	USDHKD	USDHUF	USDILS
USDJPY	USDMXN	USDNOK	USDPLN	USDRUB	USDSEK
USDSGD	USDZAR	XAGUSD	XAUUSD	ZARJPY	

Examples

The following examples demonstrate some common practices for requesting CFD data.

Example 1: SMA Crossing

The following strategy purchases ETFs when their fast-moving average crosses over the slow-moving average. In live mode, the strategy purchases their CFD equivalents.

C#

```
public class EtfCfdSwapMovingAverageCrossAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Request the data to feed into indicators and trade.
        // In live mode, request CFD data for trading.
        // CFD is more useful in non-trading hours to reduce slippage when market prices jump at market opening.
        var tickers = new[] {"SPY", "BND", "GLD", "QQQ"};
        var securityType = LiveMode ? SecurityType.Cfd : SecurityType.Equity;
        foreach (var ticker in tickers)
        {
            // Create fast and slow-moving average indicators to trade SAM cross trend-following strategy.
            var equity = AddEquity(ticker, Resolution.Daily);
            equity["smaFast"] = SMA(equity.Symbol, 50);
            equity["smaSlow"] = SMA(equity.Symbol, 200);
            equity["targetVehicle"] = LiveMode ? AddCfd(ticker, Resolution.Daily, market:
Market.InteractiveBrokers).Symbol : equity.Symbol;

            // Create an schduled event to plot the prices.
            // We need to create a symbol representing the US Equity market
            // because market hours for IB CFDs are not modeled
            var market = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
            Schedule.On(DateRules.EveryDay(market), TimeRules.Midnight, ()=>
            {
                foreach (var security in Securities.Values)
                {
                    Plot(security.Symbol.Value, "EOD", security.Price);
                }
            });
        }
    }

    public override void OnData(Slice data)
    {
        foreach (var security in Securities.Values)
        {
            var targetVehicle = security["targetVehicle"] as Symbol;
            // Trade the SMA cross on the updated SMA indicator signals.
            // If fast MA is above slow MA, the price is in an uptrend, and we buy.
            if ((SimpleMovingAverage)security["smaFast"]  > (SimpleMovingAverage)security["smaSlow"])
            {
                SetHoldings(targetVehicle, 0.25m);
            }
            // Else, the price is in a downtrend, and we are short.
            else
            {
                SetHoldings(targetVehicle, 0m);
            }
        }
    }
}
```

CFD

Handling Data

Introduction

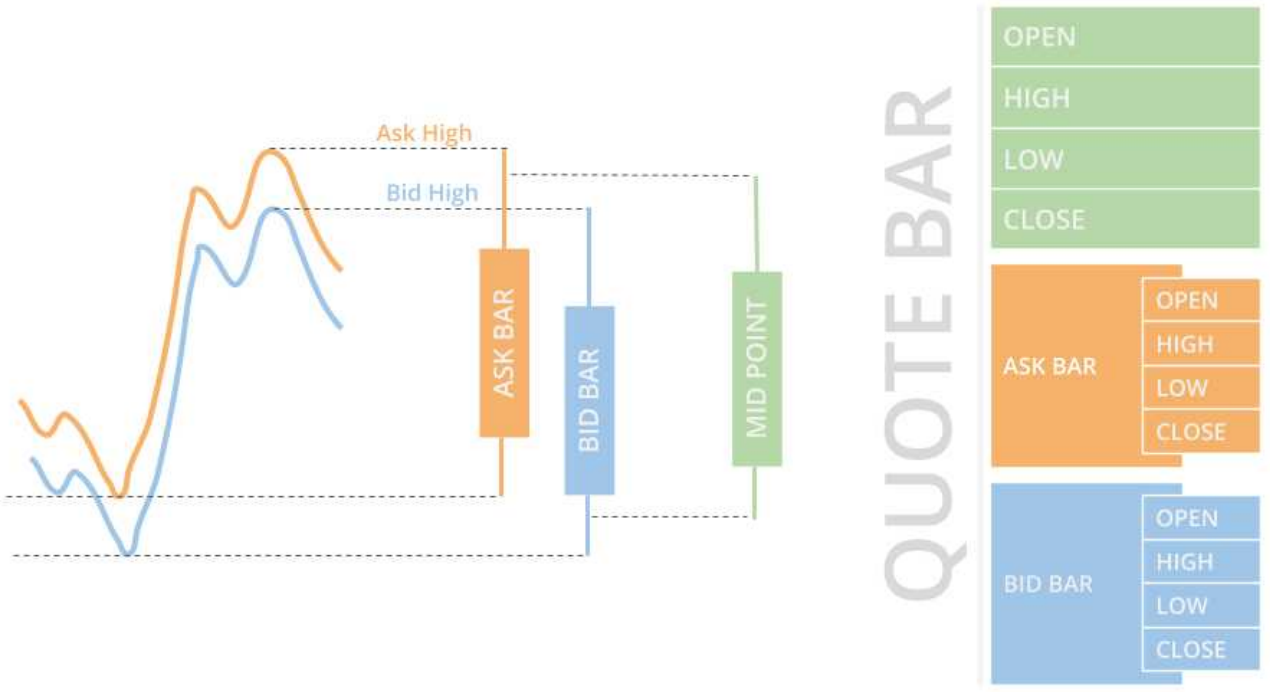
LEAN passes the data you request to the `OnData` method so you can make trading decisions. The default `OnData` method accepts a `Slice` object, but you can define additional `OnData` methods that accept different data types. For example, if you define an `OnData` method that accepts a `QuoteBar` argument, it only receives `QuoteBar` objects. The `Slice` object that the `OnData` method receives groups all the data together at a single moment in time. To access the `Slice` outside of the `OnData` method, use the `CurrentSlice` property of your algorithm.

All the data formats use `DataDictionary` objects to group data by `Symbol` and provide easy access to information. The plural of the type denotes the collection of objects. For instance, the `QuoteBars` `DataDictionary` is made up of `QuoteBar` objects. To access individual data points in the dictionary, you can index the dictionary with the CFD ticker or `Symbol` , but we recommend you use the `Symbol` .

To view the resolutions that are available for CFD data, see Resolutions .

Quotes

`QuoteBar` objects are bars that consolidate NBBO quotes from the exchanges. They contain the open, high, low, and close prices of the bid and ask. The `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` object are the mean of the respective bid and ask prices. If the bid or ask portion of the `QuoteBar` has no data, the `Open` , `High` , `Low` , and `Close` properties of the `QuoteBar` copy the values of either the `Bid` or `Ask` instead of taking their mean.



To get the `QuoteBar` objects in the `Slice` , index the `QuoteBars` property of the `Slice` with the CFD `Symbol` . If the CFD doesn't actively get quotes or you are in the same time step as when you added the CFD subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your CFD before you index the `Slice` with the CFD `Symbol`

.

```C#
public override void OnData(Slice slice)
{
    // Check if the symbol is contained in QuoteBars object
    if (slice.QuoteBars.ContainsKey(_symbol))
    {
        // Obtain the mapped QuoteBar of the symbol
        var quoteBar = slice.QuoteBars[_symbol];
    }
}
```

You can also iterate through the `QuoteBars` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `QuoteBar` objects.

```C#
public override void OnData(Slice slice)
{
    // Iterate all received Symbol-QuoteBar key-value pairs
    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        var askPrice = quoteBar.Ask.Close;
    }
}
```

`QuoteBar` objects let LEAN incorporate spread costs into your simulated trade fills to make backtest results more realistic.

`QuoteBar` objects have the following properties:

Ticks

`Tick` objects represent a single trade or quote at a moment in time. A trade tick is a record of a transaction for the CFD. A quote tick is an offer to buy or sell the CFD at a specific price.

Trade ticks have a non-zero value for the `Quantity` and `Price` properties, but they have a zero value for the `BidPrice` , `BidSize` , `AskPrice` , and `AskSize` properties. Quote ticks have non-zero values for `BidPrice` and `BidSize` properties or have non-zero values for `AskPrice` and `AskSize` properties. To check if a tick is a trade or a quote, use the `TickType` property.

In backtests, LEAN groups ticks into one millisecond buckets. In live trading, LEAN groups ticks into ~70-millisecond buckets. To get the `Tick` objects in the `Slice` , index the `Ticks` property of the `Slice` with a `Symbol` . If the CFD doesn't actively trade or you are in the same time step as when you added the CFD subscription, the `Slice` may not contain data for your `Symbol` . To avoid issues, check if the `Slice` contains data for your CFD before you index the `Slice` with the CFD `Symbol` .

```C#
public override void OnData(Slice slice)
{
    if (slice.Ticks.ContainsKey(_symbol))
    {
        var ticks = slice.Ticks[_symbol];
        foreach (var tick in ticks)
        {
            var price = tick.Price;
        }
    }
}
```

You can also iterate through the `Ticks` dictionary. The keys of the dictionary are the `Symbol` objects and the values are the `List<Tick>` objects.

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Ticks)
    {
        var symbol = kvp.Key;
        var ticks = kvp.Value;
        foreach (var tick in ticks)
        {
            var price = tick.Price;
        }
    }
}
```

Tick data is raw and unfiltered, so it can contain bad ticks that skew your trade results. For example, some ticks come from dark pools, which aren't tradable. We recommend you only use tick data if you understand the risks and are able to perform your own online tick filtering.

`Tick` objects have the following properties:

Examples

The following examples demonstrate common practices for handling CFD data.

Example 1: Buy And Hold

The following algorithm is a buy-and-hold strategy on the DE30EUR CFD contract. We only place orders when quote data is available to avoid stale fills. Also, we set the algorithm time zone to Berlin and the account currency to Euros for a more convenient comparison.

```C#
public class CfdExampleAlgorithm : QCAlgorithm
{
    private Symbol _dax, _sgx;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Set the timezone as Berlin to conveniently compare the data.
        SetTimeZone(TimeZones.Berlin);
        // Set the account currency to EUR to trade the DAX CFD.
        SetAccountCurrency("EUR", 10000);
        // Request the CFD data to trade.
        _dax = AddCfd("DE30EUR").Symbol;
        _sgx = AddCfd("SG30SGD").Symbol;
    }

    public override void OnData(Slice slice)
    {
        // Trade based on updated data; CFDs only have quote data.
        if (slice.QuoteBars.TryGetValue(_dax, out var _))
        {
            // Buy and hold DAX CFD.
            if (!Portfolio[_dax].IsLong)
            {
                SetHoldings(_dax, 0.5m);
            }
        }
    }
}
```

Example 2: Global Indices

The following algorithm buys and holds various Index CFDs that trade in different market hours and currencies during their opening hours.

C#

```
public class CfdExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Seed the price of each asset with its last known price to
        // avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Let's select CFD contracts that trade in different market
        // hours so the algorithm is always invested.
        foreach (var ticker in new[] { "DE30EUR", "SG30SGD", "US30USD" })
        {
            // Add the CFD.
            var symbol = AddCfd(ticker).Symbol;
            // Set scheduled events to hold each CFD contract during
            // their regular trading hours.
            // Buy after market open.
            Schedule.On(
                DateRules.EveryDay(symbol),
                TimeRules.AfterMarketOpen(symbol, 1),
                () => SetHoldings(symbol, 0.3m)
            );
            // Liquidate before market close.
            Schedule.On(
                DateRules.EveryDay(symbol),
                TimeRules.BeforeMarketClose(symbol, 1),
                () => Liquidate(symbol)
            );
        }
    }
}
```

CFD

Market Hours

Market Hours

Interactive Brokers

Introduction

This page shows the trading hours, holidays, and time zone of the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CFD market:

Weekday	**Time (America/New York)**
Monday	09:30:00 to 16:00:00
Tuesday	09:30:00 to 16:00:00
Wednesday	09:30:00 to 16:00:00
Thursday	09:30:00 to 16:00:00
Friday	09:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21

2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01

2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

The following table shows the early closes for the CFD market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/New York)**
1999-11-26	13:00:00
2000-07-03	13:00:00

2000-11-24	13:00:00
2001-07-03	13:00:00
2001-11-23	13:00:00
2001-12-24	13:00:00
2002-07-05	13:00:00
2002-11-29	13:00:00
2002-12-24	13:00:00
2003-07-03	13:00:00
2003-11-28	13:00:00
2003-12-24	13:00:00
2003-12-26	13:00:00
2004-11-26	13:00:00
2005-11-25	13:00:00
2006-07-03	13:00:00
2006-11-24	13:00:00
2007-07-03	13:00:00
2007-11-23	13:00:00
2007-12-24	13:00:00
2008-07-03	13:00:00
2008-11-28	13:00:00
2008-12-24	13:00:00
2009-11-27	13:00:00
2009-12-24	13:00:00
2010-11-26	13:00:00
2011-11-25	13:00:00
2012-07-03	13:00:00
2012-11-23	13:00:00
2012-12-24	13:00:00

2013-07-03	13:00:00
2013-11-29	13:00:00
2013-12-24	13:00:00
2014-07-03	13:00:00
2014-11-28	13:00:00
2014-12-24	13:00:00
2015-11-27	13:00:00
2015-12-24	13:00:00
2016-11-25	13:00:00
2017-07-03	13:00:00
2017-11-24	13:00:00
2017-12-24	13:00:00
2018-07-03	13:00:00
2018-11-23	13:00:00
2018-12-24	13:00:00
2019-07-03	13:00:00
2019-11-29	13:00:00
2019-12-24	13:00:00
2020-11-27	13:00:00
2020-12-24	13:00:00
2021-11-26	13:00:00
2022-11-25	13:00:00
2023-07-03	13:00:00
2023-11-24	13:00:00
2024-07-03	13:00:00
2024-11-29	13:00:00
2024-12-24	13:00:00
2025-07-03	13:00:00

2025-11-28	13:00:00
2025-12-24	13:00:00
2026-11-27	13:00:00
2026-12-24	13:00:00

Late Opens

There are no days with late opens.

Time Zone

The CFD market trades in the following time zones:

- `America/New York`
- `UTC`

Assets With Other Hours

The following table shows the contracts that have different trading periods than the overall CFD market:

Symbol	Name
AUDCAD	AUD/CAD
AUDCHF	AUD/CHF
AUDCNH	AUD/CNH
AUDHKD	AUD/HKD
AUDJPY	AUD/JPY
AUDNZD	AUD/NZD
AUDSGD	AUD/SGD
AUDUSD	AUD/USD
AUDZAR	AUD/ZAR
CADCHF	CAD/CHF
CADCNH	CAD/CNH
CADHKD	CAD/HKD
CADJPY	CAD/JPY
CHFCNH	CHF/CNH
CHFCZK	CHF/CZK
CHFDKK	CHF/DKK

CHFHUF	CHF/HUF
CHFJPY	CHF/JPY
CHFNOK	CHF/NOK
CHFPLN	CHF/PLN
CHFSEK	CHF/SEK
CHFZAR	CHF/ZAR
CNHHKD	CNH/HKD
CNHJPY	CNH/JPY
DKKJPY	DKK/JPY
DKKNOK	DKK/NOK
DKKSEK	DKK/SEK
EURAUD	EUR/AUD
EURCAD	EUR/CAD
EURCHF	EUR/CHF
EURCNH	EUR/CNH
EURCZK	EUR/CZK
EURDKK	EUR/DKK
EURGBP	EUR/GBP
EURHKD	EUR/HKD
EURHUF	EUR/HUF
EURILS	EUR/ILS
EURJPY	EUR/JPY
EURMXN	EUR/MXN
EURNOK	EUR/NOK
EURNZD	EUR/NZD
EURPLN	EUR/PLN
EURRUB	EUR/RUB
EURSEK	EUR/SEK

EURSGD	EUR/SGD
EURUSD	EUR-USD
EURZAR	EUR/ZAR
GBPAUD	GBP/AUD
GBPCAD	GBP/CAD
GBPCHF	GBP/CHF
GBPCNH	GBP/CNH
GBPCZK	GBP/CZK
GBPDKK	GBP/DKK
GBPHKD	GBP/HKD
GBPHUF	GBP/HUF
GBPJPY	GBP/JPY
GBPMXN	GBP/MXN
GBPNOK	GBP/NOK
GBPNZD	GBP/NZD
GBPPLN	GBP/PLN
GBPSEK	GBP/SEK
GBPSGD	GBP/SGD
GBPUSD	GBP/USD
GBPZAR	GBP/ZAR
HKDJPY	HKD/JPY
IBAU200	Australia 200
IBCH20	Switzerland 20
IBDE40	Germany 40
IBES35	Spain 35
IBEU50	Europe 50
IBFR40	France 40
IBGB100	UK 100

IBHK50	Hong Kong 50
IBJP225	Japan 225
IBNL25	Netherlands 25
IBUS30	US 30
IBUS500	US 500
IBUST100	US Tech 100
MXNJPY	MXN/JPY
NOKJPY	NOK/JPY
NOKSEK	NOK/SEK
NZDCAD	NZD/CAD
NZDCHF	NZD/CHF
NZDJPY	NZD/JPY
NZDUSD	NZD/USD
SEKJPY	SEK/JPY
SGDCNH	SGD/CNH
SGDJPY	SGD/JPY
USDCAD	USD/CAD
USDCHF	USD/CHF
USDCNH	USD/CNH
USDCZK	USD/CZK
USDDKK	USD/DKK
USDHKD	USD/HKD
USDHUF	USD/HUF
USDILS	USD/ILS
USDJPY	USD/JPY
USDMXN	USD/MXN
USDNOK	USD/NOK
USDPLN	USD/PLN

USDRUB	USD/RUB
USDSEK	USD/SEK
USDSGD	USD/SGD
USDZAR	USD/ZAR
XAGUSD	London Silver
XAUUSD	London Gold
ZARJPY	ZAR/JPY

Interactive Brokers

AUDCAD

Introduction

This page shows the trading hours, holidays, and time zone of the AUD/CAD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the AUD/CAD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The AUD/CAD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

AUDCHF

Introduction

This page shows the trading hours, holidays, and time zone of the AUD/CHF contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the AUD/CHF contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The AUD/CHF contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

AUDCNH

Introduction

This page shows the trading hours, holidays, and time zone of the AUD/CNH contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the AUD/CNH contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The AUD/CNH contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

AUDHKD

Introduction

This page shows the trading hours, holidays, and time zone of the AUD/HKD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the AUD/HKD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The AUD/HKD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

AUDJPY

Introduction

This page shows the trading hours, holidays, and time zone of the AUD/JPY contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the AUD/JPY contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The AUD/JPY contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

AUDNZD

Introduction

This page shows the trading hours, holidays, and time zone of the AUD/NZD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the AUD/NZD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The AUD/NZD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

AUDSGD

Introduction

This page shows the trading hours, holidays, and time zone of the AUD/SGD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the AUD/SGD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The AUD/SGD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

AUDUSD

Introduction

This page shows the trading hours, holidays, and time zone of the AUD/USD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the AUD/USD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The AUD/USD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

AUDZAR

Introduction

This page shows the trading hours, holidays, and time zone of the AUD/ZAR contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the AUD/ZAR contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The AUD/ZAR contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

CADCHF

Introduction

This page shows the trading hours, holidays, and time zone of the CAD/CHF contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CAD/CHF contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CAD/CHF contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

CADCNH

Introduction

This page shows the trading hours, holidays, and time zone of the CAD/CNH contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CAD/CNH contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CAD/CNH contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

CADHKD

Introduction

This page shows the trading hours, holidays, and time zone of the CAD/HKD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CAD/HKD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CAD/HKD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

CADJPY

Introduction

This page shows the trading hours, holidays, and time zone of the CAD/JPY contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CAD/JPY contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CAD/JPY contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

CHFCNH

Introduction

This page shows the trading hours, holidays, and time zone of the CHF/CNH contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CHF/CNH contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CHF/CNH contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

CHFCZK

Introduction

This page shows the trading hours, holidays, and time zone of the CHF/CZK contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CHF/CZK contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CHF/CZK contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

CHFDKK

Introduction

This page shows the trading hours, holidays, and time zone of the CHF/DKK contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CHF/DKK contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CHF/DKK contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

CHFHUF

Introduction

This page shows the trading hours, holidays, and time zone of the CHF/HUF contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CHF/HUF contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CHF/HUF contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

CHFJPY

Introduction

This page shows the trading hours, holidays, and time zone of the CHF/JPY contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CHF/JPY contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CHF/JPY contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

CHFNOK

Introduction

This page shows the trading hours, holidays, and time zone of the CHF/NOK contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CHF/NOK contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CHF/NOK contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

CHFPLN

Introduction

This page shows the trading hours, holidays, and time zone of the CHF/PLN contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CHF/PLN contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CHF/PLN contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

CHFSEK

Introduction

This page shows the trading hours, holidays, and time zone of the CHF/SEK contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CHF/SEK contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CHF/SEK contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

CHFZAR

Introduction

This page shows the trading hours, holidays, and time zone of the CHF/ZAR contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CHF/ZAR contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CHF/ZAR contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

CNHHKD

Introduction

This page shows the trading hours, holidays, and time zone of the CNH/HKD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CNH/HKD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CNH/HKD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

CNHJPY

Introduction

This page shows the trading hours, holidays, and time zone of the CNH/JPY contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CNH/JPY contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CNH/JPY contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

DKKJPY

Introduction

This page shows the trading hours, holidays, and time zone of the DKK/JPY contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DKK/JPY contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DKK/JPY contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

DKKNOK

Introduction

This page shows the trading hours, holidays, and time zone of the DKK/NOK contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DKK/NOK contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DKK/NOK contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

DKKSEK

Introduction

This page shows the trading hours, holidays, and time zone of the DKK/SEK contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the DKK/SEK contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The DKK/SEK contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

EURAUD

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/AUD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/AUD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR/AUD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

EURCAD

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/CAD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/CAD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR/CAD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

EURCHF

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/CHF contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/CHF contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR/CHF contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

EURCNH

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/CNH contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/CNH contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR/CNH contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

EURCZK

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/CZK contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/CZK contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR/CZK contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

EURDKK

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/DKK contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/DKK contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR/DKK contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

EURGBP

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/GBP contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/GBP contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR/GBP contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

EURHKD

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/HKD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/HKD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR/HKD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

EURHUF

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/HUF contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/HUF contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR/HUF contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

EURILS

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/ILS contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/ILS contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR/ILS contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

EURJPY

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/JPY contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/JPY contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR/JPY contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

EURMXN

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/MXN contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/MXN contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR/MXN contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

EURNOK

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/NOK contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/NOK contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR/NOK contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

EURNZD

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/NZD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/NZD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR/NZD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

EURPLN

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/PLN contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/PLN contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR/PLN contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

EURRUB

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/RUB contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/RUB contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR/RUB contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

EURSEK

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/SEK contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/SEK contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR/SEK contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

EURSGD

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/SGD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/SGD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR/SGD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

EURUSD

Introduction

This page shows the trading hours, holidays, and time zone of the EUR-USD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR-USD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR-USD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

EURZAR

Introduction

This page shows the trading hours, holidays, and time zone of the EUR/ZAR contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the EUR/ZAR contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The EUR/ZAR contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

GBPAUD

Introduction

This page shows the trading hours, holidays, and time zone of the GBP/AUD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GBP/AUD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The GBP/AUD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

GBPCAD

Introduction

This page shows the trading hours, holidays, and time zone of the GBP/CAD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GBP/CAD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The GBP/CAD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

GBPCHF

Introduction

This page shows the trading hours, holidays, and time zone of the GBP/CHF contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GBP/CHF contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The GBP/CHF contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

GBPCNH

Introduction

This page shows the trading hours, holidays, and time zone of the GBP/CNH contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GBP/CNH contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The GBP/CNH contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

GBPCZK

Introduction

This page shows the trading hours, holidays, and time zone of the GBP/CZK contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GBP/CZK contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The GBP/CZK contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

GBPDKK

Introduction

This page shows the trading hours, holidays, and time zone of the GBP/DKK contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GBP/DKK contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The GBP/DKK contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

GBPHKD

Introduction

This page shows the trading hours, holidays, and time zone of the GBP/HKD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GBP/HKD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The GBP/HKD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

GBPHUF

Introduction

This page shows the trading hours, holidays, and time zone of the GBP/HUF contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GBP/HUF contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The GBP/HUF contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

GBPJPY

Introduction

This page shows the trading hours, holidays, and time zone of the GBP/JPY contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GBP/JPY contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The GBP/JPY contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

GBPMXN

Introduction

This page shows the trading hours, holidays, and time zone of the GBP/MXN contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GBP/MXN contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The GBP/MXN contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

GBPNOK

Introduction

This page shows the trading hours, holidays, and time zone of the GBP/NOK contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GBP/NOK contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The GBP/NOK contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

GBPNZD

Introduction

This page shows the trading hours, holidays, and time zone of the GBP/NZD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GBP/NZD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The GBP/NZD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

GBPPLN

Introduction

This page shows the trading hours, holidays, and time zone of the GBP/PLN contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GBP/PLN contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The GBP/PLN contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

GBPSEK

Introduction

This page shows the trading hours, holidays, and time zone of the GBP/SEK contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GBP/SEK contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The GBP/SEK contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

GBPSGD

Introduction

This page shows the trading hours, holidays, and time zone of the GBP/SGD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GBP/SGD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The GBP/SGD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

GBPUSD

Introduction

This page shows the trading hours, holidays, and time zone of the GBP/USD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GBP/USD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The GBP/USD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

GBPZAR

Introduction

This page shows the trading hours, holidays, and time zone of the GBP/ZAR contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the GBP/ZAR contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The GBP/ZAR contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

HKDJPY

Introduction

This page shows the trading hours, holidays, and time zone of the HKD/JPY contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the HKD/JPY contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The HKD/JPY contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

IBAU200

Introduction

This page shows the trading hours, holidays, and time zone of the Australia 200 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Australia 200 contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	22:50:00 to 24:00:00
Monday	00:00:00 to 05:30:00, 06:10:00 to 21:00:00, 22:50:00 to 24:00:00
Tuesday	00:00:00 to 05:30:00, 06:10:00 to 21:00:00, 22:50:00 to 24:00:00
Wednesday	00:00:00 to 05:30:00, 06:10:00 to 21:00:00, 22:50:00 to 24:00:00
Thursday	00:00:00 to 05:30:00, 06:10:00 to 21:00:00, 22:50:00 to 24:00:00
Friday	00:00:00 to 05:30:00, 06:10:00 to 21:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Australia 200 contract in the CFD market trades in the `UTC` time zone.

Interactive Brokers

IBCH20

Introduction

This page shows the trading hours, holidays, and time zone of the Switzerland 20 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Switzerland 20 contract in the CFD market:

Weekday	**Time (UTC)**
Monday	07:00:00 to 21:00:00
Tuesday	07:00:00 to 21:00:00
Wednesday	07:00:00 to 21:00:00
Thursday	07:00:00 to 21:00:00
Friday	07:00:00 to 21:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Switzerland 20 contract in the CFD market trades in the `UTC` time zone.

Interactive Brokers

IBDE40

Introduction

This page shows the trading hours, holidays, and time zone of the Germany 40 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Germany 40 contract in the CFD market:

Weekday	**Time (UTC)**
Monday	01:15:00 to 21:00:00
Tuesday	01:15:00 to 21:00:00
Wednesday	01:15:00 to 21:00:00
Thursday	01:15:00 to 21:00:00
Friday	01:15:00 to 21:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Germany 40 contract in the CFD market trades in the `UTC` time zone.

Interactive Brokers

IBES35

Introduction

This page shows the trading hours, holidays, and time zone of the Spain 35 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Spain 35 contract in the CFD market:

Weekday	**Time (UTC)**
Monday	07:00:00 to 19:00:00
Tuesday	07:00:00 to 19:00:00
Wednesday	07:00:00 to 19:00:00
Thursday	07:00:00 to 19:00:00
Friday	07:00:00 to 19:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Spain 35 contract in the CFD market trades in the `UTC` time zone.

Interactive Brokers

IBEU50

Introduction

This page shows the trading hours, holidays, and time zone of the Europe 50 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Europe 50 contract in the CFD market:

Weekday	**Time (UTC)**
Monday	01:15:00 to 21:00:00
Tuesday	01:15:00 to 21:00:00
Wednesday	01:15:00 to 21:00:00
Thursday	01:15:00 to 21:00:00
Friday	01:15:00 to 21:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Europe 50 contract in the CFD market trades in the `UTC` time zone.

Interactive Brokers

IBFR40

Introduction

This page shows the trading hours, holidays, and time zone of the France 40 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the France 40 contract in the CFD market:

Weekday	**Time (UTC)**
Monday	07:00:00 to 21:00:00
Tuesday	07:00:00 to 21:00:00
Wednesday	07:00:00 to 21:00:00
Thursday	07:00:00 to 21:00:00
Friday	07:00:00 to 21:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The France 40 contract in the CFD market trades in the `UTC` time zone.

Interactive Brokers

IBGB100

Introduction

This page shows the trading hours, holidays, and time zone of the UK 100 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the UK 100 contract in the CFD market:

Weekday	**Time (UTC)**
Monday	07:00:00 to 21:00:00
Tuesday	07:00:00 to 21:00:00
Wednesday	07:00:00 to 21:00:00
Thursday	07:00:00 to 21:00:00
Friday	07:00:00 to 21:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The UK 100 contract in the CFD market trades in the `UTC` time zone.

Interactive Brokers

IBHK50

Introduction

This page shows the trading hours, holidays, and time zone of the Hong Kong 50 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Hong Kong 50 contract in the CFD market:

Weekday	**Time (UTC)**
Monday	01:15:00 to 04:00:00, 05:00:00 to 08:30:00, 09:15:00 to 19:00:00
Tuesday	01:15:00 to 04:00:00, 05:00:00 to 08:30:00, 09:15:00 to 19:00:00
Wednesday	01:15:00 to 04:00:00, 05:00:00 to 08:30:00, 09:15:00 to 19:00:00
Thursday	01:15:00 to 04:00:00, 05:00:00 to 08:30:00, 09:15:00 to 19:00:00
Friday	01:15:00 to 04:00:00, 05:00:00 to 08:30:00, 09:15:00 to 19:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Hong Kong 50 contract in the CFD market trades in the `UTC` time zone.

Interactive Brokers

IBJP225

Introduction

This page shows the trading hours, holidays, and time zone of the Japan 225 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Japan 225 contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:00:00 to 24:00:00
Monday	00:00:00 to 22:00:00, 23:00:00 to 24:00:00
Tuesday	00:00:00 to 22:00:00, 23:00:00 to 24:00:00
Wednesday	00:00:00 to 22:00:00, 23:00:00 to 24:00:00
Thursday	00:00:00 to 22:00:00, 23:00:00 to 24:00:00
Friday	00:00:00 to 22:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Japan 225 contract in the CFD market trades in the `UTC` time zone.

Interactive Brokers

IBNL25

Introduction

This page shows the trading hours, holidays, and time zone of the Netherlands 25 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Netherlands 25 contract in the CFD market:

Weekday	**Time (UTC)**
Monday	07:00:00 to 21:00:00
Tuesday	07:00:00 to 21:00:00
Wednesday	07:00:00 to 21:00:00
Thursday	07:00:00 to 21:00:00
Friday	07:00:00 to 21:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12
2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01

2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05
2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20

2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Netherlands 25 contract in the CFD market trades in the `UTC` time zone.

Interactive Brokers

IBUS30

Introduction

This page shows the trading hours, holidays, and time zone of the US 30 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US 30 contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	22:0:00 to 24:00:00
Monday	00:00:00 to 21:00:00, 22:0:00 to 24:00:00
Tuesday	00:00:00 to 21:00:00, 22:0:00 to 24:00:00
Wednesday	00:00:00 to 21:00:00, 22:0:00 to 24:00:00
Thursday	00:00:00 to 21:00:00, 22:0:00 to 24:00:00
Friday	00:00:00 to 21:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The US 30 contract in the CFD market trades in the `UTC` time zone.

Interactive Brokers

IBUS500

Introduction

This page shows the trading hours, holidays, and time zone of the US 500 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US 500 contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	22:0:00 to 24:00:00
Monday	00:00:00 to 21:00:00, 22:0:00 to 24:00:00
Tuesday	00:00:00 to 21:00:00, 22:0:00 to 24:00:00
Wednesday	00:00:00 to 21:00:00, 22:0:00 to 24:00:00
Thursday	00:00:00 to 21:00:00, 22:0:00 to 24:00:00
Friday	00:00:00 to 21:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The US 500 contract in the CFD market trades in the `UTC` time zone.

Interactive Brokers

IBUST100

Introduction

This page shows the trading hours, holidays, and time zone of the US Tech 100 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US Tech 100 contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	22:0:00 to 24:00:00
Monday	00:00:00 to 21:00:00, 22:0:00 to 24:00:00
Tuesday	00:00:00 to 21:00:00, 22:0:00 to 24:00:00
Wednesday	00:00:00 to 21:00:00, 22:0:00 to 24:00:00
Thursday	00:00:00 to 21:00:00, 22:0:00 to 24:00:00
Friday	00:00:00 to 21:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The US Tech 100 contract in the CFD market trades in the `UTC` time zone.

Interactive Brokers

MXNJPY

Introduction

This page shows the trading hours, holidays, and time zone of the MXN/JPY contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the MXN/JPY contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The MXN/JPY contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

NOKJPY

Introduction

This page shows the trading hours, holidays, and time zone of the NOK/JPY contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NOK/JPY contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The NOK/JPY contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

NOKSEK

Introduction

This page shows the trading hours, holidays, and time zone of the NOK/SEK contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NOK/SEK contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The NOK/SEK contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

NZDCAD

Introduction

This page shows the trading hours, holidays, and time zone of the NZD/CAD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NZD/CAD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The NZD/CAD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

NZDCHF

Introduction

This page shows the trading hours, holidays, and time zone of the NZD/CHF contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NZD/CHF contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The NZD/CHF contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

NZDJPY

Introduction

This page shows the trading hours, holidays, and time zone of the NZD/JPY contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NZD/JPY contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The NZD/JPY contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

NZDUSD

Introduction

This page shows the trading hours, holidays, and time zone of the NZD/USD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the NZD/USD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The NZD/USD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

SEKJPY

Introduction

This page shows the trading hours, holidays, and time zone of the SEK/JPY contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the SEK/JPY contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The SEK/JPY contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

SGDCNH

Introduction

This page shows the trading hours, holidays, and time zone of the SGD/CNH contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the SGD/CNH contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The SGD/CNH contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

SGDJPY

Introduction

This page shows the trading hours, holidays, and time zone of the SGD/JPY contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the SGD/JPY contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The SGD/JPY contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

USDCAD

Introduction

This page shows the trading hours, holidays, and time zone of the USD/CAD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the USD/CAD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The USD/CAD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

USDCHF

Introduction

This page shows the trading hours, holidays, and time zone of the USD/CHF contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the USD/CHF contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The USD/CHF contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

USDCNH

Introduction

This page shows the trading hours, holidays, and time zone of the USD/CNH contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the USD/CNH contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The USD/CNH contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

USDCZK

Introduction

This page shows the trading hours, holidays, and time zone of the USD/CZK contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the USD/CZK contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The USD/CZK contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

USDDKK

Introduction

This page shows the trading hours, holidays, and time zone of the USD/DKK contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the USD/DKK contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The USD/DKK contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

USDHKD

Introduction

This page shows the trading hours, holidays, and time zone of the USD/HKD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the USD/HKD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The USD/HKD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

USDHUF

Introduction

This page shows the trading hours, holidays, and time zone of the USD/HUF contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the USD/HUF contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The USD/HUF contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

USDILS

Introduction

This page shows the trading hours, holidays, and time zone of the USD/ILS contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the USD/ILS contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The USD/ILS contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

USDJPY

Introduction

This page shows the trading hours, holidays, and time zone of the USD/JPY contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the USD/JPY contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The USD/JPY contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

USDMXN

Introduction

This page shows the trading hours, holidays, and time zone of the USD/MXN contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the USD/MXN contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The USD/MXN contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

USDNOK

Introduction

This page shows the trading hours, holidays, and time zone of the USD/NOK contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the USD/NOK contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The USD/NOK contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

USDPLN

Introduction

This page shows the trading hours, holidays, and time zone of the USD/PLN contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the USD/PLN contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The USD/PLN contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

USDRUB

Introduction

This page shows the trading hours, holidays, and time zone of the USD/RUB contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the USD/RUB contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The USD/RUB contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

USDSEK

Introduction

This page shows the trading hours, holidays, and time zone of the USD/SEK contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the USD/SEK contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The USD/SEK contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

USDSGD

Introduction

This page shows the trading hours, holidays, and time zone of the USD/SGD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the USD/SGD contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The USD/SGD contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

USDZAR

Introduction

This page shows the trading hours, holidays, and time zone of the USD/ZAR contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the USD/ZAR contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The USD/ZAR contract in the CFD market trades in the `America/New York` time zone.

Interactive Brokers

XAGUSD

Introduction

This page shows the trading hours, holidays, and time zone of the London Silver contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the London Silver contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	22:00:00 to 24:00:00
Monday	00:00:00 to 21:00:00, 22:00:00 to 24:00:00
Tuesday	00:00:00 to 21:00:00, 22:00:00 to 24:00:00
Wednesday	00:00:00 to 21:00:00, 22:00:00 to 24:00:00
Thursday	00:00:00 to 21:00:00, 22:00:00 to 24:00:00
Friday	00:00:00 to 21:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The London Silver contract in the CFD market trades in the `UTC` time zone.

Interactive Brokers

XAUUSD

Introduction

This page shows the trading hours, holidays, and time zone of the London Gold contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the London Gold contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	22:00:00 to 24:00:00
Monday	00:00:00 to 21:00:00, 22:00:00 to 24:00:00
Tuesday	00:00:00 to 21:00:00, 22:00:00 to 24:00:00
Wednesday	00:00:00 to 21:00:00, 22:00:00 to 24:00:00
Thursday	00:00:00 to 21:00:00, 22:00:00 to 24:00:00
Friday	00:00:00 to 21:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The London Gold contract in the CFD market trades in the `UTC` time zone.

Interactive Brokers

ZARJPY

Introduction

This page shows the trading hours, holidays, and time zone of the ZAR/JPY contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the ZAR/JPY contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:15:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)				
1998-01-01	1998-01-19	1998-02-16	1998-04-10	1998-05-25
1998-07-03	1998-09-07	1998-11-26	1998-12-25	1999-01-01
1999-01-18	1999-02-15	1999-04-02	1999-05-31	1999-07-05
1999-09-06	1999-11-25	1999-12-24	2000-01-17	2000-02-21
2000-04-21	2000-05-29	2000-07-04	2000-09-04	2000-11-23
2000-12-25	2001-01-01	2001-01-15	2001-02-19	2001-04-13
2001-05-28	2001-07-04	2001-09-03	2001-09-11	2001-09-12

2001-09-13	2001-09-14	2001-11-22	2001-12-25	2002-01-01
2002-01-21	2002-02-18	2002-03-29	2002-05-27	2002-07-04
2002-09-02	2002-11-28	2002-12-25	2003-01-01	2003-01-20
2003-02-17	2003-04-18	2003-05-26	2003-07-04	2003-09-01
2003-11-27	2003-12-25	2004-01-01	2004-01-19	2004-02-16
2004-04-09	2004-05-31	2004-06-11	2004-07-05	2004-09-06
2004-11-25	2004-12-24	2005-01-17	2005-02-21	2005-03-25
2005-05-30	2005-07-04	2005-09-05	2005-11-24	2005-12-26
2006-01-02	2006-01-16	2006-02-20	2006-04-14	2006-05-29
2006-07-04	2006-09-04	2006-11-23	2006-12-25	2007-01-01
2007-01-02	2007-01-15	2007-02-19	2007-04-06	2007-05-28
2007-07-04	2007-09-03	2007-11-22	2007-12-25	2008-01-01
2008-01-21	2008-02-18	2008-03-21	2008-05-26	2008-07-04
2008-09-01	2008-11-27	2008-12-25	2009-01-01	2009-01-19
2009-02-16	2009-04-10	2009-05-25	2009-07-03	2009-09-07
2009-11-26	2009-12-25	2010-01-01	2010-01-18	2010-02-15
2010-04-02	2010-05-31	2010-07-05	2010-09-06	2010-11-25
2010-12-24	2011-01-01	2011-01-17	2011-02-21	2011-04-22
2011-05-30	2011-07-04	2011-09-05	2011-11-24	2011-12-26
2012-01-02	2012-01-16	2012-02-20	2012-04-06	2012-05-28
2012-07-04	2012-09-03	2012-10-29	2012-10-30	2012-11-22
2012-12-25	2013-01-01	2013-01-21	2013-02-18	2013-03-29
2013-05-27	2013-07-04	2013-09-02	2013-11-28	2013-12-25
2014-01-01	2014-01-20	2014-02-17	2014-04-18	2014-05-26
2014-07-04	2014-09-01	2014-11-27	2014-12-25	2015-01-01
2015-01-19	2015-02-16	2015-04-03	2015-05-25	2015-07-03
2015-09-07	2015-11-26	2015-12-25	2016-01-01	2016-01-18
2016-02-15	2016-03-25	2016-05-30	2016-07-04	2016-09-05

2016-11-24	2016-12-26	2017-01-02	2017-01-16	2017-02-20
2017-04-14	2017-05-29	2017-07-04	2017-09-04	2017-11-23
2017-12-25	2018-01-01	2018-01-15	2018-02-19	2018-03-30
2018-05-28	2018-07-04	2018-09-03	2018-11-22	2018-12-05
2018-12-25	2019-01-01	2019-01-21	2019-02-18	2019-04-19
2019-05-27	2019-07-04	2019-09-02	2019-11-28	2019-12-25
2020-01-01	2020-01-20	2020-02-17	2020-04-10	2020-05-25
2020-07-03	2020-09-07	2020-11-26	2020-12-25	2021-01-01
2021-01-18	2021-02-15	2021-04-02	2021-05-31	2021-07-05
2021-09-06	2021-11-25	2021-12-24	2022-01-01	2022-01-17
2022-02-21	2022-04-15	2022-05-30	2022-06-20	2022-07-04
2022-09-05	2022-11-24	2022-12-26	2023-01-02	2023-01-16
2023-02-20	2023-04-07	2023-05-29	2023-06-19	2023-07-04
2023-09-04	2023-11-23	2023-12-25	2024-01-01	2024-01-15
2024-02-19	2024-03-29	2024-05-27	2024-06-19	2024-07-04
2024-09-02	2024-11-28	2024-12-25	2025-01-01	2025-01-20
2025-02-17	2025-04-18	2025-06-19	2025-07-04	2025-09-01
2025-11-27	2025-12-25	2026-01-01	2026-01-19	2026-02-16
2026-04-03	2026-05-25	2026-06-19	2026-07-03	2026-09-07
2026-11-26	2026-12-25			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The ZAR/JPY contract in the CFD market trades in the `America/New York` time zone.

Market Hours

Oanda

Introduction

This page shows the trading hours, holidays, and time zone of the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CFD market:

Weekday	**Time (America/New York)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 24:00:00
Tuesday	00:00:00 to 24:00:00
Wednesday	00:00:00 to 24:00:00
Thursday	00:00:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)			
2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

The following table shows the early closes for the CFD market:

Date (*yyyy-mm-dd*)	**Time Of Market Close (America/New York)**
2018-12-31	17:00:00

Late Opens

There are no days with late opens.

Time Zone

The CFD market trades in the following time zones:

- `America/Chicago`
- `America/New York`
- `Asia/Hong Kong`
- `Asia/Singapore`
- `Australia/Sydney`
- `Europe/Amsterdam`
- `Europe/Berlin`
- `Europe/London`
- `Europe/Paris`
- `Europe/Zurich`
- `UTC`

Assets With Other Hours

The following table shows the contracts that have different trading periods than the overall CFD market:

Symbol	Name
AU200AUD	Australia 200
BCOUSD	Brent Crude Oil
CH20CHF	Swiss 20
CORNUSD	CORN/USD
DE10YBEUR	Bund
DE30EUR	Germany 30
EU50EUR	Europe 50
FR40EUR	France 40
HK33HKD	Hong Kong 33
JP225USD	Japan 225
NAS100USD	US Nas 100
NATGASUSD	Natural Gas
NL25EUR	Netherlands 25
SG30SGD	Singapore 30
SOYBNUSD	Soybeans
SPX500USD	US SPX 500

SUGARUSD	Sugar
UK100GBP	UK 100
UK10YBGBP	UK 10Y Gilt
US2000USD	US Russ 2000
US30USD	US Wall St 30
USB02YUSD	US 2Y T-Note
USB05YUSD	US 5Y T-Note
USB10YUSD	US 10Y T-Note
USB30YUSD	US T-Bond
WHEATUSD	Wheat
WTICOUSD	West Texas Oil
XAGAUD	Silver/AUD
XAGCAD	Silver/CAD
XAGCHF	Silver/CHF
XAGEUR	Silver/EUR
XAGGBP	Silver/GBP
XAGHKD	Silver/HKD
XAGJPY	Silver/JPY
XAGNZD	Silver/NZD
XAGSGD	Silver/SGD
XAGUSD	London Silver
XAUAUD	Gold/AUD
XAUCAD	Gold/CAD
XAUCHF	Gold/CHF
XAUEUR	Gold/EUR
XAUGBP	Gold/GBP
XAUHKD	Gold/HKD
XAUJPY	Gold/JPY

XAUNZD	Gold/NZD
XAUSGD	Gold/SGD
XAUUSD	London Gold
XAUXAG	Gold/Silver
XCUUSD	Copper
XPDUSD	Palladium
XPTUSD	Platinum

Oanda

AU200AUD

Introduction

This page shows the trading hours, holidays, and time zone of the Australia 200 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Australia 200 contract in the CFD market:

Weekday	**Time (Australia/Sydney)**
Monday	09:50:00 to 16:30:00, 17:10:00 to 24:00:00
Tuesday	00:00:00 to 08:00:00, 09:50:00 to 16:30:00, 17:10:00 to 24:00:00
Wednesday	00:00:00 to 08:00:00, 09:50:00 to 16:30:00, 17:10:00 to 24:00:00
Thursday	00:00:00 to 08:00:00, 09:50:00 to 16:30:00, 17:10:00 to 24:00:00
Friday	00:00:00 to 08:00:00, 09:50:00 to 16:30:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Australia 200 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2004-04-09	2004-12-26	2004-12-27	2005-03-25	2005-12-25
2005-12-26	2006-04-14	2006-12-25	2007-01-26	2007-04-06
2007-12-25	2008-03-21	2008-04-25	2008-12-25	2008-12-26
2009-04-10	2009-12-25	2010-01-01	2010-04-02	2010-12-26
2010-12-27	2011-04-22	2011-04-25	2011-12-25	2011-12-26
2011-12-27	2012-01-02	2012-01-26	2012-04-06	2012-04-09
2012-04-25	2012-12-24	2012-12-25	2012-12-31	2013-01-28
2013-03-29	2013-12-25	2014-01-27	2014-04-18	2014-04-25
2014-12-25	2014-12-26	2015-04-03	2015-12-24	2015-12-25
2015-12-31	2016-01-01	2016-03-25	2016-12-23	2016-12-25
2016-12-26	2016-12-30	2017-04-14	2017-12-25	2018-03-30
2018-12-24	2018-12-25	2018-12-31	2019-01-01	2019-01-28
2019-04-19	2019-04-22	2019-04-25	2019-06-10	2019-12-25
2019-12-26	2020-01-01	2020-01-27	2020-04-10	2020-04-13
2020-04-25	2020-06-08	2020-12-25	2020-12-26	2020-12-28
2021-01-01	2021-01-26	2021-04-02	2021-04-05	2021-04-26
2021-06-14	2021-12-25	2021-12-26	2021-12-27	2021-12-28
2022-01-03	2022-01-26	2022-04-15	2022-04-18	2022-04-25
2022-06-13	2022-12-27	2023-01-26	2023-04-07	2023-04-10
2023-04-25	2023-06-12	2023-12-25	2023-12-26	2024-01-01
2024-01-26	2024-03-29	2024-04-01	2024-04-25	2024-06-10
2024-12-25	2024-12-26	2025-01-01	2025-01-27	2025-04-18
2025-04-21	2025-04-25	2025-06-09	2025-12-25	2025-12-26
2026-01-01	2026-01-26	2026-04-03	2026-04-06	2026-04-25
2026-06-08	2026-12-25	2026-12-26	2026-12-28	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Australia 200 contract in the CFD market trades in the `Australia/Sydney` time zone.

Oanda

BCOUSD

Introduction

This page shows the trading hours, holidays, and time zone of the Brent Crude Oil contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Brent Crude Oil contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	20:00:00 to 24:00:00
Monday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Tuesday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Wednesday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Thursday	00:00:00 to 18:00:00, 20:00:00 to 24:00:00
Friday	00:00:00 to 18:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Brent Crude Oil contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2017-12-25	2018-01-01	2018-03-30	2018-12-25
2019-01-01	2019-12-25	2020-01-01	2020-04-10	2020-12-25
2021-01-01	2021-04-02	2022-04-15		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Brent Crude Oil contract in the CFD market trades in the `America/New York` time zone.

Oanda

CH20CHF

Introduction

This page shows the trading hours, holidays, and time zone of the Swiss 20 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Swiss 20 contract in the CFD market:

Weekday	**Time (Europe/Zurich)**
Monday	08:00:00 to 22:00:00
Tuesday	08:00:00 to 22:00:00
Wednesday	08:00:00 to 22:00:00
Thursday	08:00:00 to 22:00:00
Friday	08:00:00 to 22:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Swiss 20 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-04-14	2017-04-17	2017-05-01	2017-05-25
2017-06-05	2017-08-01	2017-12-25	2017-12-26	2018-01-01
2018-03-30	2018-04-02	2018-05-01	2018-05-10	2018-05-21
2018-08-01	2018-12-25	2018-12-26	2019-01-01	2019-04-19
2019-04-22	2019-05-01	2019-05-30	2019-06-10	2019-08-01
2019-12-25	2019-12-26	2020-01-01	2020-04-10	2020-04-13
2020-05-01	2020-06-01	2020-12-25	2021-01-01	2021-04-02
2021-04-05	2021-05-13	2021-05-24	2021-12-24	2021-12-31
2022-04-15	2022-04-18	2022-05-26	2022-06-06	2022-08-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Swiss 20 contract in the CFD market trades in the `Europe/Zurich` time zone.

Oanda

CORNUSD

Introduction

This page shows the trading hours, holidays, and time zone of the CORN/USD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the CORN/USD contract in the CFD market:

Weekday	**Time (America/Chicago)**
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Tuesday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Wednesday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Thursday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Friday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CORN/USD contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-16	2017-02-20	2017-04-14	2017-05-29
2017-07-04	2017-09-04	2017-11-23	2017-12-25	2018-01-01
2018-01-15	2018-02-19	2018-03-30	2018-05-28	2018-07-04
2018-09-03	2018-11-22	2018-12-25	2019-01-01	2019-01-21
2019-02-18	2019-04-19	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2019-12-25	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-07-03	2020-09-07	2020-11-26
2020-12-25	2021-01-01	2021-01-18	2021-02-15	2021-04-02
2021-05-31	2021-07-05	2021-09-06	2021-11-25	2021-12-24
2021-12-31	2022-01-17	2022-02-21	2022-04-15	2022-05-13
2022-07-04	2022-09-05	2022-11-24		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The CORN/USD contract in the CFD market trades in the `America/Chicago` time zone.

Oanda

DE10YBEUR

Introduction

This page shows the trading hours, holidays, and time zone of the Bund contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Bund contract in the CFD market:

Weekday	**Time (Europe/Berlin)**
Monday	01:15:00 to 22:00:00
Tuesday	01:15:00 to 22:00:00
Wednesday	01:15:00 to 22:00:00
Thursday	01:15:00 to 22:00:00
Friday	01:15:00 to 22:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Bund contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2017-04-17	2017-05-01	2017-06-05	2017-10-03
2017-10-31	2017-12-25	2017-12-26	2018-01-01	2018-03-30
2018-04-02	2018-05-01	2018-05-21	2018-10-03	2018-10-31
2018-12-25	2018-12-26	2019-01-01	2019-04-19	2019-04-22
2019-05-01	2019-06-10	2019-10-03	2019-10-31	2019-12-25
2019-12-26	2020-01-01	2020-04-10	2020-04-13	2020-05-01
2020-06-01	2020-12-25	2021-01-01	2021-04-02	2021-04-05
2021-05-01	2021-05-24	2021-12-24	2021-12-31	2022-04-15
2022-04-18	2022-05-01	2022-06-06	2022-10-03	2022-10-31

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Bund contract in the CFD market trades in the `Europe/Berlin` time zone.

Oanda

DE30EUR

Introduction

This page shows the trading hours, holidays, and time zone of the Germany 30 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Germany 30 contract in the CFD market:

Weekday	**Time (Europe/Berlin)**
Monday	01:15:00 to 22:00:00
Tuesday	01:15:00 to 22:00:00
Wednesday	01:15:00 to 22:00:00
Thursday	01:15:00 to 22:00:00
Friday	01:15:00 to 22:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Germany 30 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2017-04-17	2017-05-01	2017-06-05	2017-10-03
2017-10-31	2017-12-25	2017-12-26	2018-01-01	2018-03-30
2018-04-02	2018-05-01	2018-05-21	2018-10-03	2018-10-31
2018-12-25	2018-12-26	2019-01-01	2019-04-19	2019-04-22
2019-05-01	2019-06-10	2019-10-03	2019-10-31	2019-12-25
2019-12-26	2020-01-01	2020-04-10	2020-04-13	2020-05-01
2020-06-01	2020-12-25	2021-01-01	2021-04-02	2021-04-05
2021-05-01	2021-05-24	2021-12-24	2021-12-31	2022-04-15
2022-04-18	2022-05-01	2022-06-06	2022-10-03	2022-10-31

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Germany 30 contract in the CFD market trades in the `Europe/Berlin` time zone.

Oanda

EU50EUR

Introduction

This page shows the trading hours, holidays, and time zone of the Europe 50 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Europe 50 contract in the CFD market:

Weekday	**Time (Europe/Berlin)**
Monday	01:15:00 to 22:00:00
Tuesday	01:15:00 to 22:00:00
Wednesday	01:15:00 to 22:00:00
Thursday	01:15:00 to 22:00:00
Friday	01:15:00 to 22:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Europe 50 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2003-04-18	2003-04-21	2003-05-01	2003-12-25	2003-12-26
2004-01-01	2004-04-09	2004-04-12	2004-12-26	2005-03-25
2005-03-28	2005-05-01	2005-12-25	2005-12-26	2006-01-01
2006-04-14	2006-04-17	2006-05-01	2006-12-25	2006-12-26
2007-01-01	2007-04-06	2007-04-09	2007-05-01	2007-12-25
2007-12-26	2008-01-01	2008-03-21	2008-03-24	2008-05-01
2008-12-25	2008-12-26	2009-01-01	2009-04-10	2009-04-13
2009-05-01	2009-12-25	2010-01-01	2010-04-02	2010-04-05
2010-12-26	2011-04-22	2011-04-25	2011-05-01	2011-12-25
2011-12-26	2012-01-01	2012-04-06	2012-04-09	2012-05-01
2012-12-25	2013-01-01	2013-03-29	2013-04-01	2013-05-01
2013-12-25	2013-12-26	2014-01-01	2014-04-18	2014-04-21
2014-05-01	2014-12-25	2014-12-26	2015-01-01	2015-04-03
2015-04-06	2015-05-01	2015-12-25	2016-01-01	2016-03-25
2016-03-28	2016-05-01	2016-12-25	2016-12-26	2017-01-01
2017-04-14	2017-04-17	2017-05-01	2017-12-25	2017-12-26
2018-01-01	2018-03-30	2018-04-02	2018-05-01	2018-12-25
2018-12-26	2019-01-01	2019-04-19	2019-04-22	2019-05-01
2019-12-25	2019-12-26	2020-01-01	2020-04-10	2020-04-13
2020-05-01	2020-12-25	2020-12-26	2021-01-01	2021-04-02
2021-04-05	2021-05-01	2021-12-25	2021-12-26	2022-01-01
2022-04-15	2022-04-18	2022-05-01	2023-04-07	2023-04-10
2023-05-01	2023-12-25	2023-12-26	2024-01-01	2024-03-29
2024-04-01	2024-05-01	2024-12-25	2024-12-26	2025-01-01
2025-04-18	2025-04-21	2025-05-01	2025-12-25	2025-12-26
2026-01-01	2026-04-03	2026-04-06	2026-05-01	2026-12-25
2026-12-26	2027-01-01			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Europe 50 contract in the CFD market trades in the `Europe/Berlin` time zone.

Oanda

FR40EUR

Introduction

This page shows the trading hours, holidays, and time zone of the France 40 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the France 40 contract in the CFD market:

Weekday	**Time (Europe/Paris)**
Monday	08:00:00 to 22:00:00
Tuesday	08:00:00 to 22:00:00
Wednesday	08:00:00 to 22:00:00
Thursday	08:00:00 to 22:00:00
Friday	08:00:00 to 22:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the France 40 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2003-04-18	2003-04-21	2003-05-01	2003-12-25	2003-12-26
2004-01-01	2004-04-09	2004-04-12	2004-12-26	2005-03-25
2005-03-28	2005-05-01	2005-12-25	2005-12-26	2006-01-01
2006-04-14	2006-04-17	2006-05-01	2006-12-25	2006-12-26
2007-01-01	2007-04-06	2007-04-09	2007-05-01	2007-12-25
2007-12-26	2008-01-01	2008-03-21	2008-03-24	2008-05-01
2008-12-25	2008-12-26	2009-01-01	2009-04-10	2009-04-13
2009-05-01	2009-12-25	2010-01-01	2010-04-02	2010-04-05
2010-12-26	2011-04-22	2011-04-25	2011-05-01	2011-12-25
2011-12-26	2012-01-01	2012-04-06	2012-04-09	2012-05-01
2012-12-25	2013-01-01	2013-03-29	2013-04-01	2013-05-01
2013-12-25	2013-12-26	2014-01-01	2014-04-18	2014-04-21
2014-05-01	2014-12-25	2014-12-26	2015-01-01	2015-04-03
2015-04-06	2015-05-01	2015-12-25	2016-01-01	2016-03-25
2016-03-28	2016-05-01	2016-12-25	2016-12-26	2017-01-01
2017-04-14	2017-04-17	2017-05-01	2017-12-25	2017-12-26
2018-01-01	2018-03-30	2018-04-02	2018-05-01	2018-12-25
2018-12-26	2019-01-01	2019-04-19	2019-04-22	2019-05-01
2019-12-25	2019-12-26	2020-01-01	2020-04-10	2020-04-13
2020-05-01	2020-12-25	2020-12-26	2021-01-01	2021-04-02
2021-04-05	2021-05-01	2021-12-25	2021-12-26	2022-01-01
2022-04-15	2022-04-18	2022-05-01	2023-04-07	2023-04-10
2023-05-01	2023-12-25	2023-12-26	2024-01-01	2024-03-29
2024-04-01	2024-05-01	2024-12-25	2024-12-26	2025-01-01
2025-04-18	2025-04-21	2025-05-01	2025-12-25	2025-12-26
2026-01-01	2026-04-03	2026-04-06	2026-05-01	2026-12-25
2026-12-26	2027-01-01			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The France 40 contract in the CFD market trades in the `Europe/Paris` time zone.

Oanda

HK33HKD

Introduction

This page shows the trading hours, holidays, and time zone of the Hong Kong 33 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Hong Kong 33 contract in the CFD market:

Weekday	**Time (Asia/Hong Kong)**
Monday	09:15:00 to 12:00:00, 13:00:00 to 16:30:00, 17:15:00 to 24:00:00
Tuesday	00:00:00 to 03:00:00, 09:15:00 to 12:00:00, 13:00:00 to 16:30:00, 17:15:00 to 24:00:00
Wednesday	00:00:00 to 03:00:00, 09:15:00 to 12:00:00, 13:00:00 to 16:30:00, 17:15:00 to 24:00:00
Thursday	00:00:00 to 03:00:00, 09:15:00 to 12:00:00, 13:00:00 to 16:30:00, 17:15:00 to 24:00:00
Friday	00:00:00 to 03:00:00, 09:15:00 to 12:00:00, 13:00:00 to 16:30:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Hong Kong 33 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2003-01-31	2003-02-02	2003-02-03	2003-04-18	2003-04-21
2003-05-01	2003-05-08	2003-06-04	2003-07-01	2003-09-12
2003-10-01	2003-12-25	2003-12-26	2004-01-01	2004-01-22
2004-01-23	2004-04-05	2004-04-09	2004-04-12	2004-05-26
2004-06-22	2004-07-01	2004-09-29	2004-10-01	2004-10-22
2004-12-26	2004-12-27	2005-02-09	2005-02-10	2005-02-11
2005-03-25	2005-03-28	2005-04-05	2005-05-02	2005-05-16
2005-07-01	2005-09-19	2005-10-11	2005-12-25	2005-12-26

2005-12-27	2006-01-02	2006-01-29	2006-01-30	2006-01-31
2006-04-05	2006-04-14	2006-04-17	2006-05-01	2006-05-05
2006-05-31	2006-10-02	2006-10-30	2006-12-25	2006-12-26
2007-01-01	2007-02-18	2007-02-19	2007-02-20	2007-04-05
2007-04-06	2007-04-09	2007-05-01	2007-05-24	2007-06-19
2007-07-02	2007-09-26	2007-10-01	2007-10-19	2007-12-25
2007-12-26	2008-01-01	2008-02-07	2008-02-08	2008-03-21
2008-03-24	2008-04-04	2008-05-01	2008-05-12	2008-06-09
2008-07-01	2008-08-06	2008-08-22	2008-09-15	2008-10-01
2008-10-07	2008-12-25	2008-12-26	2009-01-01	2009-01-26
2009-01-27	2009-01-28	2009-04-10	2009-04-13	2009-05-01
2009-05-28	2009-07-01	2009-10-01	2009-10-04	2009-10-26
2009-12-25	2010-01-01	2010-02-14	2010-02-15	2010-02-16
2010-04-02	2010-04-05	2010-04-06	2010-05-21	2010-06-16
2010-07-01	2010-09-23	2010-10-01	2010-12-26	2010-12-27
2011-02-03	2011-02-04	2011-04-05	2011-04-22	2011-04-25
2011-05-02	2011-05-10	2011-06-06	2011-07-01	2011-09-13
2011-09-29	2011-10-05	2011-12-25	2011-12-26	2011-12-27
2012-01-02	2012-01-23	2012-01-24	2012-01-25	2012-12-25
2013-01-01	2013-02-10	2013-02-11	2013-02-12	2013-02-13
2013-03-29	2013-04-01	2013-04-04	2013-05-01	2013-05-17
2013-06-12	2013-07-01	2013-08-14	2013-09-20	2013-10-01
2013-10-14	2013-12-25	2013-12-26	2014-01-01	2014-01-31
2014-02-02	2014-02-03	2014-04-18	2014-04-21	2014-05-01
2014-05-06	2014-06-02	2014-07-01	2014-09-09	2014-10-01
2014-10-02	2014-12-25	2014-12-26	2015-01-01	2015-02-19
2015-02-20	2015-04-03	2015-04-06	2015-05-01	2015-05-25
2015-12-24	2015-12-25	2015-12-31	2016-01-01	2016-02-08

2016-02-09	2016-03-25	2016-07-01	2016-09-16	2016-12-25
2016-12-26	2017-01-29	2017-01-30	2017-04-14	2017-12-25
2018-02-16	2018-02-18	2018-03-30	2018-12-24	2018-12-25
2018-12-31	2019-01-01	2019-02-05	2019-02-06	2019-02-07
2019-04-05	2019-04-19	2019-04-22	2019-05-01	2019-05-13
2019-06-07	2019-07-01	2019-09-14	2019-10-01	2019-10-07
2019-12-25	2019-12-26	2020-01-01	2020-01-25	2020-01-26
2020-01-27	2020-01-28	2020-04-04	2020-04-10	2020-04-13
2020-04-30	2020-05-01	2020-06-25	2020-07-01	2020-10-01
2020-10-02	2020-10-26	2020-12-25	2020-12-26	2021-01-01
2021-02-12	2021-02-13	2021-02-14	2021-02-15	2021-04-02
2021-04-05	2021-04-06	2021-05-01	2021-05-19	2021-06-14
2021-07-01	2021-09-22	2021-10-01	2021-10-14	2021-12-25
2021-12-26	2021-12-27	2022-01-01	2022-02-01	2022-02-02
2022-02-03	2022-04-05	2022-04-15	2022-04-18	2022-05-02
2022-05-09	2022-06-03	2022-07-01	2022-09-11	2022-10-01
2022-10-04	2022-12-27	2023-01-22	2023-01-23	2023-01-24
2023-01-25	2023-04-05	2023-04-07	2023-04-10	2023-05-01
2023-05-26	2023-06-22	2023-07-01	2023-09-30	2023-10-02
2023-10-23	2023-12-25	2023-12-26	2024-01-01	2024-02-10
2024-02-11	2024-02-12	2024-02-13	2024-03-29	2024-04-01
2024-04-04	2024-05-01	2024-05-15	2024-06-10	2024-07-01
2024-09-18	2024-10-01	2024-10-11	2024-12-25	2024-12-26
2025-01-01	2025-01-29	2025-01-30	2025-01-31	2025-04-04
2025-04-18	2025-04-21	2025-05-01	2025-05-05	2025-05-31
2025-07-01	2025-10-01	2025-10-07	2025-10-29	2025-12-25
2025-12-26	2026-01-01	2026-02-17	2026-02-18	2026-02-19
2026-04-03	2026-04-06	2026-04-07	2026-05-01	2026-05-25

2026-06-19	2026-07-01	2026-09-26	2026-10-01	2026-10-19
2026-12-25	2026-12-26	2027-01-01		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Hong Kong 33 contract in the CFD market trades in the `Asia/Hong Kong` time zone.

Oanda

JP225USD

Introduction

This page shows the trading hours, holidays, and time zone of the Japan 225 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Japan 225 contract in the CFD market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Japan 225 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2003-04-18	2003-05-26	2003-07-04	2003-09-01	2003-11-27
2003-12-25	2004-01-01	2004-01-19	2004-02-16	2004-04-09
2004-05-31	2004-06-11	2004-09-06	2004-11-25	2004-12-24
2005-01-17	2005-02-21	2005-03-25	2005-05-30	2005-07-04
2005-12-26	2006-01-02	2006-04-14	2006-12-25	2007-01-01
2007-12-25	2008-01-01	2008-03-21	2008-12-25	2009-01-01
2009-04-10	2009-12-25	2010-01-01	2010-12-24	2011-04-22

2011-12-26	2012-01-02	2012-12-25	2013-01-01	2013-03-29
2013-12-25	2014-01-01	2014-04-18	2015-12-25	2016-01-01
2016-03-25	2017-04-14	2018-03-30	2019-01-01	2019-01-02
2019-01-03	2019-01-14	2019-02-11	2019-03-21	2019-04-29
2019-05-03	2019-05-04	2019-05-06	2019-07-15	2019-08-12
2019-09-16	2019-09-23	2019-10-14	2019-11-04	2019-11-23
2019-12-23	2019-12-31	2020-01-01	2020-01-02	2020-01-03
2020-01-13	2020-02-11	2020-03-20	2020-04-29	2020-05-04
2020-05-05	2020-07-20	2020-08-11	2020-09-21	2020-09-22
2020-10-12	2020-11-03	2020-11-23	2020-12-23	2020-12-31
2021-01-01	2021-01-02	2021-01-04	2021-01-11	2021-02-11
2021-03-20	2021-04-29	2021-05-03	2021-05-04	2021-05-05
2021-07-19	2021-08-11	2021-09-20	2021-09-23	2021-10-11
2021-11-03	2021-11-23	2021-12-23	2021-12-31	2022-01-01
2022-01-03	2022-01-10	2022-02-11	2022-03-21	2022-04-29
2022-05-03	2022-05-04	2022-05-05	2022-07-18	2022-08-11
2022-09-19	2022-09-23	2022-10-10	2022-11-03	2022-11-23
2022-12-23	2022-12-31	2023-01-03	2023-01-09	2023-02-11
2023-03-21	2023-04-29	2023-05-03	2023-05-04	2023-05-05
2023-07-17	2023-08-11	2023-09-18	2023-09-23	2023-10-09
2023-11-03	2023-11-23	2023-12-23	2024-01-01	2024-01-02
2024-01-03	2024-01-08	2024-02-12	2024-03-20	2024-04-29
2024-05-03	2024-05-04	2024-05-06	2024-07-15	2024-08-12
2024-09-16	2024-09-23	2024-10-14	2024-11-04	2024-11-23
2024-12-23	2024-12-31	2025-01-01	2025-01-02	2025-01-03
2025-01-13	2025-02-11	2025-03-20	2025-04-29	2025-05-03
2025-05-05	2025-07-21	2025-08-11	2025-09-15	2025-09-23
2025-10-13	2025-11-03	2025-11-24	2025-12-23	2025-12-31

2026-01-01	2026-01-02	2026-01-03	2026-01-12	2026-02-11
2026-03-20	2026-04-29	2026-05-04	2026-05-05	2026-07-20
2026-08-11	2026-09-21	2026-09-23	2026-10-12	2026-11-03
2026-11-23	2026-12-23	2026-12-31	2027-01-01	2027-01-02
2027-01-04				

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Japan 225 contract in the CFD market trades in the `America/Chicago` time zone.

Oanda

NAS100USD

Introduction

This page shows the trading hours, holidays, and time zone of the US Nas 100 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US Nas 100 contract in the CFD market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Nas 100 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2003-04-18	2003-07-04	2003-12-25	2004-01-01	2004-04-09
2004-06-11	2004-12-24	2005-03-25	2006-04-14	2006-12-25
2007-01-01	2007-12-25	2008-01-01	2008-03-21	2008-12-25
2009-01-01	2009-04-10	2009-12-25	2010-01-01	2010-12-24
2011-04-22	2011-12-26	2012-01-02	2012-12-25	2013-01-01
2013-03-29	2013-12-25	2014-01-01	2014-04-18	2015-12-25
2016-03-25	2017-04-14	2018-03-30	2019-01-01	2019-04-19
2019-12-25	2020-01-01	2020-04-10	2020-12-25	2021-01-01
2021-04-02	2021-12-24	2022-01-01	2022-04-15	2023-04-07
2023-12-25	2024-01-01	2024-03-29	2024-12-25	2025-01-01
2025-04-18	2025-12-25	2026-01-01	2026-04-03	2026-12-25
2027-01-01				

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The US Nas 100 contract in the CFD market trades in the `America/Chicago` time zone.

Oanda

NATGASUSD

Introduction

This page shows the trading hours, holidays, and time zone of the Natural Gas contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Natural Gas contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Natural Gas contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-16	2017-02-20	2017-04-14	2017-05-29
2017-07-04	2017-09-04	2017-11-23	2017-12-25	2018-01-01
2018-01-15	2018-02-19	2018-03-30	2018-05-28	2018-07-04
2018-09-03	2018-11-22	2018-12-29	2019-01-01	2019-01-21
2019-02-18	2019-04-19	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2019-12-25	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-09-06	2021-11-25	2021-12-24	2022-01-17	2022-02-21
2022-04-15	2022-05-30	2022-07-04	2022-09-05	2022-11-24

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Natural Gas contract in the CFD market trades in the `America/New York` time zone.

Oanda

NL25EUR

Introduction

This page shows the trading hours, holidays, and time zone of the Netherlands 25 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Netherlands 25 contract in the CFD market:

Weekday	**Time (Europe/Amsterdam)**
Monday	08:00:00 to 22:00:00
Tuesday	08:00:00 to 22:00:00
Wednesday	08:00:00 to 22:00:00
Thursday	08:00:00 to 22:00:00
Friday	08:00:00 to 22:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Netherlands 25 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2008-03-21	2008-03-24	2008-05-01	2008-12-25	2008-12-26
2009-01-01	2009-04-10	2009-04-13	2009-05-01	2009-12-25
2010-01-01	2010-04-02	2010-04-05	2010-12-26	2011-04-22
2011-04-25	2011-05-01	2011-12-25	2011-12-26	2012-01-01
2012-04-06	2012-04-09	2012-05-01	2012-12-25	2013-01-01
2013-03-29	2013-04-01	2013-05-01	2013-12-25	2013-12-26
2014-01-01	2014-04-18	2014-04-21	2014-05-01	2014-12-25
2014-12-26	2015-01-01	2015-04-03	2015-04-06	2015-05-01
2015-12-25	2016-01-01	2016-03-25	2016-03-28	2016-05-01
2016-12-25	2016-12-26	2017-01-01	2017-04-14	2017-04-17
2017-05-01	2017-12-25	2017-12-26	2018-01-01	2018-03-30
2018-04-02	2018-05-01	2018-12-25	2018-12-26	2019-01-01
2019-04-19	2019-04-22	2019-05-01	2019-12-25	2019-12-26
2020-01-01	2020-04-10	2020-04-13	2020-05-01	2020-12-25
2020-12-26	2021-01-01	2021-04-02	2021-04-05	2021-05-01
2021-12-25	2021-12-26	2022-01-01	2022-04-15	2022-04-18
2022-05-01	2023-04-07	2023-04-10	2023-05-01	2023-12-25
2023-12-26	2024-01-01	2024-03-29	2024-04-01	2024-05-01
2024-12-25	2024-12-26	2025-01-01	2025-04-18	2025-04-21
2025-05-01	2025-12-25	2025-12-26	2026-01-01	2026-04-03
2026-04-06	2026-05-01	2026-12-25	2026-12-26	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Netherlands 25 contract in the CFD market trades in the `Europe/Amsterdam` time zone.

Oanda

SG30SGD

Introduction

This page shows the trading hours, holidays, and time zone of the Singapore 30 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Singapore 30 contract in the CFD market:

Weekday	**Time (Asia/Singapore)**
Monday	08:30:00 to 17:20:00, 17:50:00 to 24:00:00
Tuesday	00:00:00 to 04:45:00, 08:30:00 to 17:20:00, 17:50:00 to 24:00:00
Wednesday	00:00:00 to 04:45:00, 08:30:00 to 17:20:00, 17:50:00 to 24:00:00
Thursday	00:00:00 to 04:45:00, 08:30:00 to 17:20:00, 17:50:00 to 24:00:00
Friday	00:00:00 to 04:45:00, 08:30:00 to 17:20:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Singapore 30 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2010-02-15	2011-02-13	2012-01-23	2013-02-11	2014-01-31
2015-02-19	2016-02-08	2017-01-02	2017-01-30	2017-04-14
2017-05-01	2017-05-10	2017-06-26	2017-08-09	2017-09-01
2017-10-18	2017-12-25	2018-01-01	2018-02-16	2018-03-30
2018-05-01	2018-08-09	2018-12-25	2019-01-01	2019-02-05
2019-04-19	2019-05-01	2019-08-09	2019-12-25	2020-01-01
2020-01-25	2020-01-27	2020-04-10	2020-05-01	2020-08-10
2020-12-25	2021-01-01	2021-02-12	2021-04-02	2021-05-01
2021-08-09	2021-12-25	2022-01-01	2022-02-01	2022-04-15
2022-05-02	2022-08-09			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Singapore 30 contract in the CFD market trades in the `Asia/Singapore` time zone.

Oanda

SOYBNUSD

Introduction

This page shows the trading hours, holidays, and time zone of the Soybeans contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Soybeans contract in the CFD market:

Weekday	**Time (America/Chicago)**
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Tuesday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Wednesday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Thursday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Friday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Soybeans contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-16	2017-02-20	2017-04-14	2017-05-29
2017-07-04	2017-09-04	2017-11-23	2017-12-25	2018-01-01
2018-01-15	2018-02-19	2018-03-30	2018-05-28	2018-07-04
2018-09-03	2018-11-22	2018-12-25	2019-01-01	2019-01-21
2019-02-18	2019-04-19	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2019-12-25	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-07-03	2020-09-07	2020-11-26
2020-12-25	2021-01-01	2021-01-18	2021-02-15	2021-04-02
2021-05-31	2021-07-05	2021-09-06	2021-11-25	2021-12-24
2021-12-31	2022-01-17	2022-02-21	2022-04-15	2022-05-13
2022-07-04	2022-09-05	2022-11-24		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Soybeans contract in the CFD market trades in the `America/Chicago` time zone.

Oanda

SPX500USD

Introduction

This page shows the trading hours, holidays, and time zone of the US SPX 500 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US SPX 500 contract in the CFD market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US SPX 500 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2003-04-18	2003-07-04	2003-12-25	2004-01-01	2004-04-09
2004-06-11	2004-12-24	2005-03-25	2006-04-14	2006-12-25
2007-01-01	2007-12-25	2008-01-01	2008-03-21	2008-12-25
2009-01-01	2009-04-10	2009-12-25	2010-01-01	2010-12-24
2011-04-22	2011-12-26	2012-01-02	2012-12-25	2013-01-01
2013-03-29	2013-12-25	2014-01-01	2014-04-18	2015-12-25
2016-03-25	2017-04-14	2018-03-30	2019-01-01	2019-04-19
2019-12-25	2020-01-01	2020-04-10	2020-12-25	2021-01-01
2021-04-02	2021-12-24	2022-01-01	2022-04-15	2023-04-07
2023-12-25	2024-01-01	2024-03-29	2024-12-25	2025-01-01
2025-04-18	2025-12-25	2026-01-01	2026-04-03	2026-12-25
2027-01-01				

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The US SPX 500 contract in the CFD market trades in the `America/Chicago` time zone.

Oanda

SUGARUSD

Introduction

This page shows the trading hours, holidays, and time zone of the Sugar contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Sugar contract in the CFD market:

Weekday	**Time (America/New York)**
Monday	03:30:00 to 13:00:00
Tuesday	03:30:00 to 13:00:00
Wednesday	03:30:00 to 13:00:00
Thursday	03:30:00 to 13:00:00
Friday	03:30:00 to 13:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Sugar contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-16	2017-02-20	2017-04-14	2017-05-29
2017-07-04	2017-09-04	2017-11-23	2017-12-25	2018-01-01
2018-01-15	2018-02-19	2018-03-30	2018-05-28	2018-07-04
2018-09-03	2018-11-22	2018-12-25	2019-01-01	2019-01-21
2019-02-18	2019-04-19	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2019-12-25	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-07-03	2020-09-07	2020-11-26
2020-12-25	2021-01-01	2021-01-18	2021-02-15	2021-04-02
2021-05-31	2021-07-05	2021-09-06	2021-11-25	2021-12-24
2022-01-17	2022-02-21	2022-04-15	2022-05-30	2022-07-04
2022-09-05	2022-11-24			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Sugar contract in the CFD market trades in the `America/New York` time zone.

Oanda

UK100GBP

Introduction

This page shows the trading hours, holidays, and time zone of the UK 100 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the UK 100 contract in the CFD market:

Weekday	**Time (Europe/London)**
Monday	01:00:00 to 21:00:00
Tuesday	01:00:00 to 21:00:00
Wednesday	01:00:00 to 21:00:00
Thursday	01:00:00 to 21:00:00
Friday	01:00:00 to 21:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the UK 100 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2003-04-18	2003-04-21	2003-05-05	2003-05-26	2003-08-25
2003-12-25	2003-12-26	2004-01-01	2004-04-09	2004-04-12
2004-05-03	2004-05-31	2004-08-30	2004-12-26	2004-12-27
2004-12-28	2005-01-03	2005-03-25	2005-03-28	2005-05-02
2005-05-30	2005-08-29	2005-12-25	2005-12-26	2005-12-27
2006-01-02	2006-04-14	2006-04-17	2006-05-01	2006-05-29
2006-08-28	2006-12-25	2006-12-26	2007-01-01	2007-04-06
2007-04-09	2007-05-07	2007-05-28	2007-08-27	2007-12-25

2007-12-26	2008-01-01	2008-03-21	2008-03-24	2008-05-05
2008-05-26	2008-08-25	2008-12-25	2008-12-26	2009-01-01
2009-04-10	2009-04-13	2009-05-04	2009-05-25	2009-08-31
2009-12-25	2009-12-28	2010-01-01	2010-04-02	2010-04-05
2010-05-03	2010-05-31	2010-08-30	2010-12-26	2010-12-27
2010-12-28	2011-01-03	2011-04-22	2011-04-25	2011-05-02
2011-05-30	2011-08-29	2011-12-25	2011-12-26	2011-12-27
2012-01-02	2012-12-25	2013-01-01	2013-03-29	2013-04-01
2013-05-06	2013-05-27	2013-08-26	2013-12-25	2013-12-26
2014-01-01	2014-04-18	2014-04-21	2014-05-05	2014-05-26
2014-08-25	2014-12-25	2014-12-26	2015-01-01	2015-04-03
2015-04-06	2015-05-04	2015-05-25	2015-08-31	2015-12-25
2016-01-01	2016-03-25	2016-12-25	2016-12-26	2017-04-14
2017-12-25	2018-03-30	2018-12-25	2019-01-01	2019-04-19
2019-04-22	2019-05-06	2019-05-27	2019-08-26	2019-12-25
2019-12-26	2020-01-01	2020-04-10	2020-04-13	2020-05-04
2020-05-25	2020-08-31	2020-12-25	2020-12-26	2020-12-28
2021-01-01	2021-04-02	2021-04-05	2021-05-03	2021-05-31
2021-08-30	2021-12-25	2021-12-26	2021-12-27	2021-12-28
2022-01-03	2022-04-15	2022-04-18	2022-05-02	2022-05-30
2022-08-29	2022-12-27	2023-04-07	2023-04-10	2023-05-01
2023-05-29	2023-08-28	2023-12-25	2023-12-26	2024-01-01
2024-03-29	2024-04-01	2024-05-06	2024-05-27	2024-08-26
2024-12-25	2024-12-26	2025-01-01	2025-04-18	2025-04-21
2025-05-05	2025-05-26	2025-08-25	2025-12-25	2025-12-26
2026-01-01	2026-04-03	2026-04-06	2026-05-04	2026-05-25
2026-08-31	2026-12-25	2026-12-26	2026-12-28	2027-01-01

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The UK 100 contract in the CFD market trades in the `Europe/London` time zone.

Oanda

UK10YBGBP

Introduction

This page shows the trading hours, holidays, and time zone of the UK 10Y Gilt contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the UK 10Y Gilt contract in the CFD market:

Weekday	**Time (Europe/London)**
Monday	08:00:00 to 18:00:00
Tuesday	08:00:00 to 18:00:00
Wednesday	08:00:00 to 18:00:00
Thursday	08:00:00 to 18:00:00
Friday	08:00:00 to 18:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the UK 10Y Gilt contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-04-14	2017-04-17	2017-05-01	2017-12-25	2017-12-26
2018-01-01	2018-03-20	2018-04-02	2018-05-01	2018-12-25
2018-12-26	2019-01-01	2019-04-19	2019-04-22	2019-05-01
2019-12-25	2019-12-26	2020-01-01	2020-04-10	2020-04-13
2020-05-01	2020-12-25	2021-01-01	2021-04-02	2021-04-05
2022-04-15	2022-04-18			

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The UK 10Y Gilt contract in the CFD market trades in the `Europe/London` time zone.

Oanda

US2000USD

Introduction

This page shows the trading hours, holidays, and time zone of the US Russ 2000 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US Russ 2000 contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 16:15:00, 16:30:00 to 17:00:00, 18:00:00 to 24:00:00
Tuesday	00:00:00 to 16:15:00, 16:30:00 to 17:00:00, 18:00:00 to 24:00:00
Wednesday	00:00:00 to 16:15:00, 16:30:00 to 17:00:00, 18:00:00 to 24:00:00
Thursday	00:00:00 to 16:15:00, 16:30:00 to 17:00:00, 18:00:00 to 24:00:00
Friday	00:00:00 to 16:15:00, 16:30:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Russ 2000 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2003-04-18	2003-12-25	2004-01-01	2004-04-09	2004-06-11
2004-12-24	2005-03-25	2006-04-14	2006-12-25	2007-01-01
2007-12-25	2008-01-01	2008-03-21	2008-12-25	2009-01-01
2009-04-10	2009-12-25	2010-01-01	2010-12-24	2011-04-22
2011-12-26	2012-01-02	2012-12-25	2013-01-01	2013-03-29
2013-12-25	2014-01-01	2014-04-18	2014-12-25	2015-01-01
2015-12-25	2016-01-01	2016-03-25	2017-04-14	2018-03-30
2019-01-01	2019-04-19	2019-12-25	2020-01-01	2020-04-10
2020-12-25	2021-01-01	2021-04-02	2021-12-24	2022-01-01
2022-04-15	2023-04-07	2023-12-25	2024-01-01	2024-03-29
2024-12-25	2025-01-01	2025-04-18	2025-12-25	2026-01-01
2026-04-03	2026-12-25	2027-01-01		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The US Russ 2000 contract in the CFD market trades in the `America/New York` time zone.

Oanda

US30USD

Introduction

This page shows the trading hours, holidays, and time zone of the US Wall St 30 contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US Wall St 30 contract in the CFD market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 15:15:00, 15:30:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US Wall St 30 contract in the CFD market:

Date (*yyyy-mm-dd*)				
2003-04-18	2003-05-26	2003-07-04	2003-09-01	2003-11-27
2003-12-25	2004-01-01	2004-01-19	2004-02-16	2004-04-09
2004-05-31	2004-06-11	2004-07-05	2004-09-06	2004-11-25
2004-12-24	2005-01-17	2005-02-21	2005-03-25	2005-05-30
2005-07-04	2005-09-05	2005-11-24	2005-12-26	2006-01-02
2006-01-16	2006-02-20	2006-04-14	2006-11-23	2006-12-25
2007-01-01	2007-01-15	2007-02-19	2007-11-22	2007-12-25
2008-01-01	2008-03-21	2008-12-25	2009-01-01	2009-04-10
2009-12-25	2010-01-01	2010-12-24	2011-04-22	2011-12-26
2012-01-02	2012-12-25	2013-01-01	2013-03-29	2013-12-25
2014-01-01	2014-04-18	2015-12-25	2016-03-25	2017-04-14
2018-03-30	2019-01-01	2019-04-19	2019-12-25	2020-01-01
2020-04-10	2020-12-25	2021-01-01	2021-04-02	2021-12-24
2022-01-01	2022-04-15	2023-04-07	2023-12-25	2024-01-01
2024-03-29	2024-12-25	2025-01-01	2025-04-18	2025-12-25
2026-01-01	2026-04-03	2026-12-25	2027-01-01	

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The US Wall St 30 contract in the CFD market trades in the `America/Chicago` time zone.

Oanda

USB02YUSD

Introduction

This page shows the trading hours, holidays, and time zone of the US 2Y T-Note contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US 2Y T-Note contract in the CFD market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US 2Y T-Note contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-16	2017-02-20	2017-04-14	2017-05-29
2017-07-04	2017-09-04	2017-11-23	2017-12-25	2018-01-01
2018-01-15	2018-02-19	2018-03-30	2018-05-28	2018-07-04
2018-09-03	2018-11-22	2018-12-25	2019-01-01	2019-01-21
2019-02-18	2019-04-19	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2019-12-25	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-07-03	2020-09-07	2020-11-26
2020-12-25	2021-01-01	2021-01-18	2021-02-15	2021-04-02
2021-05-31	2021-07-05	2021-09-06	2021-11-25	2021-12-24
2021-12-31	2022-01-17	2022-02-21	2022-04-15	2022-05-13
2022-07-04	2022-09-05	2022-11-24		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The US 2Y T-Note contract in the CFD market trades in the `America/Chicago` time zone.

Oanda

USB05YUSD

Introduction

This page shows the trading hours, holidays, and time zone of the US 5Y T-Note contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US 5Y T-Note contract in the CFD market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US 5Y T-Note contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-16	2017-02-20	2017-04-14	2017-05-29
2017-07-04	2017-09-04	2017-11-23	2017-12-25	2018-01-01
2018-01-15	2018-02-19	2018-03-30	2018-05-28	2018-07-04
2018-09-03	2018-11-22	2018-12-25	2019-01-01	2019-01-21
2019-02-18	2019-04-19	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2019-12-25	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-07-03	2020-09-07	2020-11-26
2020-12-25	2021-01-01	2021-01-18	2021-02-15	2021-04-02
2021-05-31	2021-07-05	2021-09-06	2021-11-25	2021-12-24
2021-12-31	2022-01-17	2022-02-21	2022-04-15	2022-05-13
2022-07-04	2022-09-05	2022-11-24		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The US 5Y T-Note contract in the CFD market trades in the `America/Chicago` time zone.

Oanda

USB10YUSD

Introduction

This page shows the trading hours, holidays, and time zone of the US 10Y T-Note contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US 10Y T-Note contract in the CFD market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US 10Y T-Note contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-16	2017-02-20	2017-04-14	2017-05-29
2017-07-04	2017-09-04	2017-11-23	2017-12-25	2018-01-01
2018-01-15	2018-02-19	2018-03-30	2018-05-28	2018-07-04
2018-09-03	2018-11-22	2018-12-25	2019-01-01	2019-01-21
2019-02-18	2019-04-19	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2019-12-25	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-07-03	2020-09-07	2020-11-26
2020-12-25	2021-01-01	2021-01-18	2021-02-15	2021-04-02
2021-05-31	2021-07-05	2021-09-06	2021-11-25	2021-12-24
2021-12-31	2022-01-17	2022-02-21	2022-04-15	2022-05-13
2022-07-04	2022-09-05	2022-11-24		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The US 10Y T-Note contract in the CFD market trades in the `America/Chicago` time zone.

Oanda

USB30YUSD

Introduction

This page shows the trading hours, holidays, and time zone of the US T-Bond contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the US T-Bond contract in the CFD market:

Weekday	**Time (America/Chicago)**
Sunday	17:00:00 to 24:00:00
Monday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Tuesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Wednesday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Thursday	00:00:00 to 16:00:00, 17:00:00 to 24:00:00
Friday	00:00:00 to 16:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the US T-Bond contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-16	2017-02-20	2017-04-14	2017-05-29
2017-07-04	2017-09-04	2017-11-23	2017-12-25	2018-01-01
2018-01-15	2018-02-19	2018-03-30	2018-05-28	2018-07-04
2018-09-03	2018-11-22	2018-12-25	2019-01-01	2019-01-21
2019-02-18	2019-04-19	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2019-12-25	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-07-03	2020-09-07	2020-11-26
2020-12-25	2021-01-01	2021-01-18	2021-02-15	2021-04-02
2021-05-31	2021-07-05	2021-09-06	2021-11-25	2021-12-24
2021-12-31	2022-01-17	2022-02-21	2022-04-15	2022-05-13
2022-07-04	2022-09-05	2022-11-24		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The US T-Bond contract in the CFD market trades in the `America/Chicago` time zone.

Oanda

WHEATUSD

Introduction

This page shows the trading hours, holidays, and time zone of the Wheat contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Wheat contract in the CFD market:

Weekday	**Time (America/Chicago)**
Sunday	19:00:00 to 24:00:00
Monday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Tuesday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Wednesday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Thursday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00, 19:00:00 to 24:00:00
Friday	00:00:00 to 07:45:00, 08:30:00 to 13:20:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Wheat contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-16	2017-02-20	2017-04-14	2017-05-29
2017-07-04	2017-09-04	2017-11-23	2017-12-25	2018-01-01
2018-01-15	2018-02-19	2018-03-30	2018-05-28	2018-07-04
2018-09-03	2018-11-22	2018-12-25	2019-01-01	2019-01-21
2019-02-18	2019-04-19	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2019-12-25	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-07-03	2020-09-07	2020-11-26
2020-12-25	2021-01-01	2021-01-18	2021-02-15	2021-04-02
2021-05-31	2021-07-05	2021-09-06	2021-11-25	2021-12-24
2021-12-31	2022-01-17	2022-02-21	2022-04-15	2022-05-13
2022-07-04	2022-09-05	2022-11-24		

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Wheat contract in the CFD market trades in the `America/Chicago` time zone.

Oanda

WTICOUSD

Introduction

This page shows the trading hours, holidays, and time zone of the West Texas Oil contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the West Texas Oil contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the West Texas Oil contract in the CFD market:

Date (*yyyy-mm-dd*)				
2017-01-02	2017-01-16	2017-02-20	2017-04-14	2017-05-29
2017-07-04	2017-09-04	2017-11-23	2017-12-25	2018-01-01
2018-01-15	2018-02-19	2018-03-30	2018-05-28	2018-07-04
2018-09-03	2018-11-22	2018-12-29	2019-01-01	2019-01-21
2019-02-18	2019-04-19	2019-05-27	2019-07-04	2019-09-02
2019-11-28	2019-12-25	2020-01-01	2020-01-20	2020-02-17
2020-04-10	2020-05-25	2020-09-07	2020-11-26	2020-12-25
2021-01-01	2021-01-18	2021-02-15	2021-04-02	2021-05-31
2021-09-06	2021-11-25	2021-12-24	2022-01-17	2022-02-21
2022-04-15	2022-05-30	2022-07-04	2022-09-05	2022-11-24

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The West Texas Oil contract in the CFD market trades in the `America/New York` time zone.

Oanda

XAGAUD

Introduction

This page shows the trading hours, holidays, and time zone of the Silver/AUD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Silver/AUD contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Silver/AUD contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Silver/AUD contract in the CFD market trades in the `UTC` time zone.

Oanda

XAGCAD

Introduction

This page shows the trading hours, holidays, and time zone of the Silver/CAD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Silver/CAD contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the CFD market:

Date (*yyyy-mm-dd*)			
2022-12-25	2022-12-26	2023-01-01	2023-01-02

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Silver/CAD contract in the CFD market trades in the `UTC` time zone.

Oanda

XAGCHF

Introduction

This page shows the trading hours, holidays, and time zone of the Silver/CHF contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Silver/CHF contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Silver/CHF contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Silver/CHF contract in the CFD market trades in the `UTC` time zone.

Oanda

XAGEUR

Introduction

This page shows the trading hours, holidays, and time zone of the Silver/EUR contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Silver/EUR contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Silver/EUR contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Silver/EUR contract in the CFD market trades in the `UTC` time zone.

Oanda

XAGGBP

Introduction

This page shows the trading hours, holidays, and time zone of the Silver/GBP contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Silver/GBP contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Silver/GBP contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Silver/GBP contract in the CFD market trades in the `UTC` time zone.

Oanda

XAGHKD

Introduction

This page shows the trading hours, holidays, and time zone of the Silver/HKD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Silver/HKD contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Silver/HKD contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Silver/HKD contract in the CFD market trades in the `UTC` time zone.

Oanda

XAGJPY

Introduction

This page shows the trading hours, holidays, and time zone of the Silver/JPY contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Silver/JPY contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Silver/JPY contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Silver/JPY contract in the CFD market trades in the `UTC` time zone.

Oanda

XAGNZD

Introduction

This page shows the trading hours, holidays, and time zone of the Silver/NZD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Silver/NZD contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 17:58:00, 18:03:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 17:58:00, 18:03:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 17:58:00, 18:03:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 17:58:00, 18:03:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 17:58:00, 18:03:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Silver/NZD contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Silver/NZD contract in the CFD market trades in the `UTC` time zone.

Oanda

XAGSGD

Introduction

This page shows the trading hours, holidays, and time zone of the Silver/SGD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Silver/SGD contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Silver/SGD contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Silver/SGD contract in the CFD market trades in the `UTC` time zone.

Oanda

XAGUSD

Introduction

This page shows the trading hours, holidays, and time zone of the London Silver contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the London Silver contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	18:03:00 to 24:00:00
Monday	00:00:00 to 16:58:00, 18:03:00 to 24:00:00
Tuesday	00:00:00 to 16:58:00, 18:03:00 to 24:00:00
Wednesday	00:00:00 to 16:58:00, 18:03:00 to 24:00:00
Thursday	00:00:00 to 16:58:00, 18:03:00 to 24:00:00
Friday	00:00:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the London Silver contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The London Silver contract in the CFD market trades in the `America/New York` time zone.

Oanda

XAUAUD

Introduction

This page shows the trading hours, holidays, and time zone of the Gold/AUD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold/AUD contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold/AUD contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold/AUD contract in the CFD market trades in the `UTC` time zone.

Oanda

XAUCAD

Introduction

This page shows the trading hours, holidays, and time zone of the Gold/CAD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold/CAD contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold/CAD contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold/CAD contract in the CFD market trades in the `UTC` time zone.

Oanda

XAUCHF

Introduction

This page shows the trading hours, holidays, and time zone of the Gold/CHF contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold/CHF contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold/CHF contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold/CHF contract in the CFD market trades in the `UTC` time zone.

Oanda

XAUEUR

Introduction

This page shows the trading hours, holidays, and time zone of the Gold/EUR contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold/EUR contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold/EUR contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold/EUR contract in the CFD market trades in the `UTC` time zone.

Oanda

XAUGBP

Introduction

This page shows the trading hours, holidays, and time zone of the Gold/GBP contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold/GBP contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold/GBP contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold/GBP contract in the CFD market trades in the `UTC` time zone.

Oanda

XAUHKD

Introduction

This page shows the trading hours, holidays, and time zone of the Gold/HKD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold/HKD contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold/HKD contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold/HKD contract in the CFD market trades in the `UTC` time zone.

Oanda

XAUJPY

Introduction

This page shows the trading hours, holidays, and time zone of the Gold/JPY contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold/JPY contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold/JPY contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold/JPY contract in the CFD market trades in the `UTC` time zone.

Oanda

XAUNZD

Introduction

This page shows the trading hours, holidays, and time zone of the Gold/NZD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold/NZD contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 17:58:00, 18:03:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 17:58:00, 18:03:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 17:58:00, 18:03:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 17:58:00, 18:03:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 17:58:00, 18:03:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold/NZD contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold/NZD contract in the CFD market trades in the `UTC` time zone.

Oanda

XAUSGD

Introduction

This page shows the trading hours, holidays, and time zone of the Gold/SGD contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold/SGD contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold/SGD contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold/SGD contract in the CFD market trades in the `UTC` time zone.

Oanda

XAUUSD

Introduction

This page shows the trading hours, holidays, and time zone of the London Gold contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the London Gold contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	18:03:00 to 24:00:00
Monday	00:00:00 to 16:58:00, 18:03:00 to 24:00:00
Tuesday	00:00:00 to 16:58:00, 18:03:00 to 24:00:00
Wednesday	00:00:00 to 16:58:00, 18:03:00 to 24:00:00
Thursday	00:00:00 to 16:58:00, 18:03:00 to 24:00:00
Friday	00:00:00 to 16:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the London Gold contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The London Gold contract in the CFD market trades in the `America/New York` time zone.

Oanda

XAUXAG

Introduction

This page shows the trading hours, holidays, and time zone of the Gold/Silver contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Gold/Silver contract in the CFD market:

Weekday	**Time (UTC)**
Sunday	23:10:00 to 24:00:00
Monday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Tuesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Wednesday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Thursday	00:00:00 to 21:58:00, 23:10:00 to 24:00:00
Friday	00:00:00 to 21:58:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Gold/Silver contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Gold/Silver contract in the CFD market trades in the `UTC` time zone.

Oanda

XCUUSD

Introduction

This page shows the trading hours, holidays, and time zone of the Copper contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Copper contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Copper contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Copper contract in the CFD market trades in the `America/New York` time zone.

Oanda

XPDUSD

Introduction

This page shows the trading hours, holidays, and time zone of the Palladium contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Palladium contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Palladium contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Palladium contract in the CFD market trades in the `America/New York` time zone.

Oanda

XPTUSD

Introduction

This page shows the trading hours, holidays, and time zone of the Platinum contract in the CFD market.

Pre-market Hours

Pre-market trading is not available.

Regular Trading Hours

The following table shows the regular trading hours for the Platinum contract in the CFD market:

Weekday	**Time (America/New York)**
Sunday	18:00:00 to 24:00:00
Monday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Tuesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Wednesday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Thursday	00:00:00 to 17:00:00, 18:00:00 to 24:00:00
Friday	00:00:00 to 17:00:00

Post-market Hours

Post-market trading is not available.

Holidays

The following table shows the dates of holidays for the Platinum contract in the CFD market:

Date (*yyyy-mm-dd*)
2017-04-14

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The Platinum contract in the CFD market trades in the `America/New York` time zone.

Asset Classes

India Equity

India Equity

Market Hours

Introduction

This page shows the trading hours, holidays, and time zone of the India Equity market.

Historical data for backtesting is unavailable for INDIA. In live trading, LEAN sources this data from your brokerage or a third-party data provider .

Pre-market Hours

The following table shows the pre-market hours for the India Equity market:

Weekday	**Time (Asia/Kolkata)**
Monday	09:00:00 to 09:15:00
Tuesday	09:00:00 to 09:15:00
Wednesday	09:00:00 to 09:15:00
Thursday	09:00:00 to 09:15:00
Friday	09:00:00 to 09:15:00

Regular Trading Hours

The following table shows the regular trading hours for the India Equity market:

Weekday	**Time (Asia/Kolkata)**
Monday	09:15:00 to 15:30:00
Tuesday	09:15:00 to 15:30:00
Wednesday	09:15:00 to 15:30:00
Thursday	09:15:00 to 15:30:00
Friday	09:15:00 to 15:30:00

Post-market Hours

The following table shows the post-market hours for the India Equity market:

Weekday	**Time (Asia/Kolkata)**
Monday	15:40:00 to 16:00:00
Tuesday	15:40:00 to 16:00:00
Wednesday	15:40:00 to 16:00:00
Thursday	15:40:00 to 16:00:00
Friday	15:40:00 to 16:00:00

Holidays

The following table shows the dates of holidays for the India Equity market:

Date (*yyyy-mm-dd*)				
2004-01-26	2004-04-14	2005-01-26	2005-04-14	2005-08-15
2006-01-26	2006-04-14	2006-05-01	2006-08-15	2006-10-02
2006-12-25	2007-01-26	2007-05-01	2007-08-15	2007-10-02
2007-12-25	2008-04-14	2008-05-01	2008-08-15	2008-10-02
2008-12-25	2009-01-26	2009-04-14	2009-05-01	2009-10-02
2009-12-25	2010-01-26	2010-04-14	2011-01-26	2011-03-02
2011-04-12	2011-04-14	2011-04-22	2011-08-15	2011-08-31
2011-09-01	2011-10-06	2011-10-26	2011-10-27	2011-11-07
2011-11-10	2011-12-06	2012-01-26	2012-02-20	2012-03-08
2012-04-05	2012-04-06	2012-05-01	2012-08-15	2012-08-20
2012-09-19	2012-10-02	2012-10-24	2012-11-14	2012-11-28
2012-12-25	2013-03-27	2013-03-29	2013-04-19	2013-04-24
2013-05-01	2013-08-09	2013-08-15	2013-09-09	2013-10-02
2013-10-16	2013-11-04	2013-11-15	2013-12-25	2014-02-27
2014-03-17	2014-04-08	2014-04-14	2014-04-18	2014-04-24
2014-05-01	2014-07-29	2014-08-15	2014-08-29	2014-10-02
2014-10-03	2014-10-06	2014-10-15	2014-10-24	2014-11-04
2014-11-06	2014-12-25	2015-01-26	2015-02-17	2015-03-06

2015-04-02	2015-04-03	2015-04-14	2015-05-01	2015-09-17
2015-09-25	2015-10-02	2015-10-22	2015-11-12	2015-11-25
2015-12-25	2016-01-26	2016-03-07	2016-03-24	2016-03-25
2016-04-14	2016-04-15	2016-04-19	2016-07-06	2016-08-15
2016-09-05	2016-09-13	2016-10-11	2016-10-12	2016-10-31
2016-11-14	2017-01-26	2017-02-24	2017-03-13	2017-04-04
2017-04-14	2017-05-01	2017-06-26	2017-08-15	2017-08-25
2017-10-02	2017-10-20	2017-12-25	2018-01-26	2018-02-13
2018-03-02	2018-03-29	2018-03-30	2018-05-01	2018-08-15
2018-08-22	2018-09-13	2018-09-20	2018-10-02	2018-10-18
2018-11-08	2018-11-23	2018-12-25	2019-03-04	2019-03-21
2019-04-17	2019-04-19	2019-04-29	2019-05-01	2019-06-05
2019-08-12	2019-08-15	2019-09-02	2019-09-10	2019-10-02
2019-10-08	2019-10-21	2019-10-28	2019-11-12	2019-12-25
2020-02-21	2020-03-10	2020-04-02	2020-04-06	2020-04-10
2020-04-14	2020-05-01	2020-05-25	2020-10-02	2020-11-16
2020-11-30	2020-12-25	2021-01-26	2021-03-11	2021-03-29
2021-04-02	2021-04-14	2021-04-21	2021-05-13	2021-07-21
2021-08-19	2021-09-10	2021-10-15	2021-11-05	2021-11-19
2022-01-26	2022-04-14	2022-08-15	2023-01-26	2023-04-14
2023-05-01	2023-08-15	2023-10-02	2023-12-25	2024-01-26
2024-05-01	2024-08-15	2024-10-02	2024-12-25	2025-04-14
2025-04-18	2025-05-01	2025-08-15	2025-10-02	2025-12-25
2026-01-26	2026-04-14	2026-05-01	2026-10-02	2026-12-25

Early Closes

There are no days with early closes.

Late Opens

There are no days with late opens.

Time Zone

The India Equity market trades in the `Asia/Kolkata` time zone.

Portfolio

Portfolio

Key Concepts

Introduction

The `Portfolio` object provides information about the investment state and history of all of the securities that have been in the algorithm during its lifetime.

The `Portfolio` property of the `QCAlgorithm` class is a dictionary where the keys are `Symbol` objects and the values are `SecurityHolding` objects. The `SecurityHolding` object provides information about the investment state and history of a security.

Investment Status

To get the investment status of your strategy, access the `Invested` property of a `Portfolio` object.

```
C#

// Get the investment status and return True if the portfolio is invested.
var invested = Portfolio.Invested;
```

Holdings Status

You can identify a position as long- or short-biased based on the sign of the holding quantity. Long positions have a positive quantity and short positions have a negative quantity. The `SecurityHolding` object has the following properties to describe the holding status.

```
C#

var holdings = Portfolio[_symbol];
var quantity = holdings.Quantity;
var invested = holdings.Invested;
var isLong = holdings.IsLong;
var isShort = holdings.IsShort;
```

The quantity can be lower than the minimum order size. If the quantity is less than the lot size of the asset, the `Invested` property is `false` .

Buying Power

To get the maximum buying power in your account currency you can use for a given `Symbol` and order direction, call the `GetBuyingPower` method.

```
C#

// Check the total amount of buying power available. Margin available varies by order direction and current holdings.
var availableBuyingPower = Portfolio.GetBuyingPower(_symbol, OrderDirection.Buy);
```

For more information about buying power, see Buying Power .

Cost Averaging Accounting

LEAN uses the cost averaging accounting method, which determines the cost of your holdings by taking a weighted average of all your purchase prices. For example, say you place the following buy orders:

1. Buy 10 ABC @ $10
2. Buy 5 ABC @ $11
3. Buy 20 ABC @ $14
4. Buy 3 ABC @ $9

In the preceding example, the average cost of your ABC position is (10*10 + 5*11 + 20*14 + 3*9) / (10 + 5 + 20 + 3) = 12.1579/share. In contrast, if you use the first-in, first-out (FIFO) accounting method, the cost of the first 10 shares is 10/share, not 12.1579/share.

To get the cost of your security holdings, use the `HoldingsCost` property of the SecurityHolding object . If you fill buy and sell orders, the holdings cost is the product of the holding quantity and the average price. For example, the following table shows how the average price and holdings cost changes with each buy and sell order order in a long position:

Order Quantity	Fill Price ($)	Holding Quantity	Average Price ($)	Holdings Cost ($)
2	10	2	(2 * 10) / 2 = 10	2 * 10 = 20
-1	11	1	(1 * 10) / 1 = 10	1 * 10 = 10
1	12	2	(1 * 10 + 1 * 12) / 2 = 11	2 * 11 = 22
-2	13	0	0	0 * 0 = 0

The following table shows how the average price and holdings cost changes with each buy and sell order order in a short position:

Order Quantity	Fill Price ($)	Holding Quantity	Average Price ($)	Holdings Cost ($)
-2	10	-2	(-2 * 10) / -2 = 10	-2 * 10 = -20
1	11	-1	(-1 * 10) / -1 = 10	-1 * 10 = -10
-1	12	-2	(-1 * 10 + (-1) * 12) / -2 = 11	-2 * 11 = -22
2	13	0	0	0 * 0 = 0

Note that when you decrease the position size without closing the trade, the average price doesn't change because the denominator and the coefficients in the numerator of its formula are scaled by the quotient of the current holding quantity and the previous holding quantity. For instance, if the last row in the preceding table had an order quantity 1, the holding quantity would be -1 and the average price would be

$$\frac{(-1 * q) * 10 + (-1 * q) * 12}{-2 * q}$$

$$= \frac{(-1 * \frac{-1}{-2}) * 10 + (-1 * \frac{-1}{-2}) * 12}{-2 * \frac{-1}{-2}}$$

$$= \frac{\frac{-1}{2} * 10 + \frac{-1}{2} * 12}{-1}$$

$$= \frac{-5 - 6}{-1} = 11$$

Performance Statistics

The `Portfolio` object has a collection of properties that provide the combined state of holdings and the cashbook .

```C#
// Access key statistics of the portfolio using the properties of the portfolio object.
var value = Portfolio.TotalPortfolioValue;
var totalMarginUsed = Portfolio.TotalMarginUsed;
var marginRemaining = Portfolio.MarginRemaining;
var totalNetProfit = Portfolio.TotalNetProfit;
var totalUnrealisedProfit = Portfolio.TotalUnrealisedProfit;
var totalFees = Portfolio.TotalFees;
var cash = Portfolio.Cash;
```

Properties

The `Portfolio` object has the following properties:

Examples

The following examples demonstrate common practices for handling the portfolio.

Example 1: Martingale

The following algorithm trades a Martingale strategy on AAPL. The trend indicated by an EMA indicator follows the trend with a 1*pershare**stop* – *loss**and*2 take-profit level. Yet, if the last trade is lost, it will double the next trading size to recover the last loss.

C#

```
public class PortfolioExampleAlgorithm : QCAlgorithm
{
    private Symbol _aapl;
    private ExponentialMovingAverage _ema;
    private int _nextQuantity = 1;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request AAPL's data to trade.
        _aapl = AddEquity("AAPL").Symbol;

        // Create an EMA indicator to generate trading data.
        _ema = EMA(_aapl, 20, Resolution.Daily);
        // Warm up the indicator to provide an immediate trade signal.
        WarmUpIndicator(_aapl, _ema, Resolution.Daily);
    }

    public override void OnData(Slice slice)
    {
        // Ride on the AAPL trend indicated by the EMA indicator.
        if (!Portfolio.Invested)
        {
            if (Securities[_aapl].Price >= _ema)
            {
                MarketOrder(_aapl, _nextQuantity);
            }
            else
            {
                MarketOrder(_aapl, -_nextQuantity);
            }
        }
        else
        {
            var holding = Portfolio[_aapl];
            // Stop loss at $1 per share.
            if (holding.UnrealizedProfit <= -1 * _nextQuantity)
            {
                Liquidate(_aapl);
                // Double the next quantity to recover this loss.
                _nextQuantity *= 2;
            }
            // Take profit at $2 per share.
            else if (holding.UnrealizedProfit >= 2)
            {
                Liquidate(_aapl);
                _nextQuantity = 1;
            }
        }
    }
}
```

Portfolio

Holdings

Introduction

The `Portfolio` is a dictionary where the key is a `Symbol` and the value is a `SecurityHolding` .

```
C#
// Create shortcut helper to the SPY SecurityHolding object from the Portfolio collection.
var securityHolding = Portfolio["SPY"];
```

Properties

`SecurityHolding` objects have the following properties:

```
C#
// Save shortcut to SecurityHolding object, and fetch quantity and invested status.
var securityHolding = Portfolio["SPY"];
var quantity = securityHolding.Quantity;
var invested = securityHolding.Invested;
```

`FutureHolding` objects also have `SettledProfit` and `UnsettledProfit` properties.

Get Total Close Profit

To get the profit of a position holding if you closed the position, call the `TotalCloseProfit` method. The value this method returns is denominated in your account currency and accounts for order fees.

```
C#
var profit = Portfolio["SPY"].TotalCloseProfit();
```

The `TotalCloseProfit` method accepts the following optional arguments:

Argument	Data Type	Description	Default Value
`includeFees`	`bool`	Whether to reduce the profit based on the estimated fee from the fee model .	`true`
`exitPrice`	`decimal?`	A hypothetical exit price to use for the profit calculation. If you don't provide a value, it uses the bid price for sell orders or the ask price for buy orders.	`null`
`entryPrice`	`decimal?`	A hypothetical exit price to use for the profit calculation. If you don't provide a value, it uses the average price of the `SecurityHolding` .	`null`
`quantity`	`decimal?`	The quantity to liquidate. If you don't provide a value, it uses the quantity of the `SecurityHolding` .	`null`

LEAN uses this method to define the `UnrealizedProfit` property.

Get Quantity Value

To get the value of a security at any quantity, call the `GetQuantityValue` method. The value this method returns is denominated in your account currency.

```
C#

// Get the quantity value at the current price
var valueAtCurrentPrice = Portfolio["SPY"].GetQuantityValue(100);

// Get the quantity value at a specific price
var valueAtSpecificPrice = Portfolio["SPY"].GetQuantityValue(100, price: 30);
```

Set Initial Holdings

When you deploy a backtest or a new paper trading algorithm, the default initial portfolio state contains $100,000 USD. However, in the Initialize method, you can define the initial portfolio state to something else. To set the initial cash balances , call `SetCash` with the currency ticker and quantity. To set multiple currencies, call the method multiple times.

```
C#

SetCash(100000);       // Set the quantity of the account currency to 100,000
SetCash("BTC", 10);    // Set the Bitcoin quantity to 10
SetCash("EUR", 10000); // Set the EUR quantity to 10,000
```

To set the initial security holdings, subscribe to the security so that it enters your portfolio , then call the `SecurityHolding.SetHoldings` method with the average purchase price and quantity.

```
C#
if (!LiveMode)
{
    Portfolio["SPY"].SetHoldings(averagePrice: 500, quantity: 100);
}
```

Examples

The following examples demonstrate common practices for setting the portfolio holdings.

Example 1: Initial Holdings

The following algorithm simulates an account with multiple initial positions and cash. Note that these initial holdings will affect the algorithm's performance.

```
C#
public class PortfolioExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Simulate initial cash in multiple currencies.
        // Note that all currencies other than the account currency will affect the performance.
        SetAccountCurrency("USD", 10000);
        SetCash("EUR", 1000);
        SetCash("JPY", 1000000);
        SetCash("HKD", 10000);

        // Add a security initializer that seeds the price of each asset with its
        // last known price. This ensures the holdings we add below will initialize
        // with a non-zero value.
        Settings.SeedInitialPrices = true;

        // Request data for some securities in the account already.
        var spy = AddEquity("SPY").Symbol;
        var aapl = AddEquity("AAPL").Symbol;
        var btcusd = AddCrypto("BTCUSD").Symbol;
        var xauusd = AddCfd("XAUUSD").Symbol;

        // Simulate that the account holds some assets initially.
        // Note that all initial holdings will affect the performance.
        Portfolio[spy].SetHoldings(averagePrice: 420m, quantity: 102m);
        Portfolio[aapl].SetHoldings(averagePrice: 102m, quantity: 25m);
        Portfolio[btcusd].SetHoldings(averagePrice: 45000m, quantity: 0.125m);
        Portfolio[xauusd].SetHoldings(averagePrice: 1000m, quantity: 100m);
    }
}
```

Portfolio

Cashbook

Introduction

The `CashBook` is a dictionary where the keys are currency tickers and the values are `Cash` objects. The `Cash` objects track the amount of each currency in the portfolio. As you buy and sell securities, LEAN credits and debits the `Cash` objects to reflect your cash holdings.

Account Currency

The default account currency is USD, but you can change it. All of the properties of the Portfolio object that return a currency value denominate the currency value in your account currency. Depending on your account currency and security subscriptions, LEAN may add internal security subscriptions to calculate the `ValueInAccountCurrency` . For example, if you only add BTCETH to your algorithm and set the account currency to USDT, LEAN adds BTCUSDT and ETHUSDT as internal feeds.

To get your account currency, use the `AccountCurrency` property.

```
C#
var accountCurrency = AccountCurrency;
```

To set your account currency, in the `Initialize` method, call the `SetAccountCurrency` method. You can only set the account currency once. LEAN ignores each additional call to `SetAccountCurrency` .

```
C#
SetAccountCurrency("EUR");
```

Settled vs Unsettled Cash

The `Portfolio` has two independent cashbooks. The `CashBook` tracks settled cash and the `UnsettledCashBook` tracks unsettled cash, which you can't spend. If you trade with a margin account, trades settle immediately so LEAN credits and debits the `CashBook` at the time of the trade. In some cases, transactions can take time to settle. For example, Equity trades in a cash account settle in T+3. Therefore, if you sell shares of stock on Monday, the transaction settles on Thursday. In contrast, Option trades settle in T+1.

```
C#
var settledCashBook = Portfolio.CashBook;
var unsettledCashBook = Portfolio.UnsettledCashBook;
```

Track Cash Balances

To get the balance of a currency in the cash book, use the `Amount` property.

```
C#
var usd = Portfolio.CashBook["USD"].Amount;
var btc = Portfolio.CashBook["BTC"].Amount;
```

To get the value of a currency in the cash book, denominated in your account currency, use the `ValueInAccountCurrency`

property.

```C#
// Return the cash book value of Ethereum in account currency (US Dollar, Euro, etc.) amount.
var ethValue = Portfolio.CashBook["ETH"].ValueInAccountCurrency;
```

Deposits and Withdrawals

In backtests, you can add and remove cash from the cash book. To add or remove cash, call the `AddAmount` method.

```C#
// Adjust the backtesting cashbook balances to add 100 USD and subtract 1.5 BTC.
var newUSDBalance = Portfolio.CashBook["USD"].AddAmount(100);
var newBTCBalance = Portfolio.CashBook["BTC"].AddAmount(-1.5m);
```

In live trading, add and withdraw cash through your brokerage account. If you adjust the cash balances in your algorithm with the `AddAmount` method, the cash balance in your algorithm will be out of sync with the cash balance in your brokerage account.

Currency Symbols

A currency symbol is a graphic that represents the currency name. For example, $ is for dollars, € for euros, and ฿ for Bitcoin. To get the symbol of a currency in the cash book, use `CurrencySymbol` property.

```C#
var usdSymbol = Portfolio.CashBook["USD"].CurrencySymbol;
```

Conversion Rates

To get the conversion rate for a currency in the cashbook to your account currency, use the `ConversionRate` property.

```C#
var eurConversionRate = Portfolio.CashBook["EUR"].ConversionRate;
```

Examples

The following examples demonstrate common practices for controlling the portfolio cashbook.

Example 1: Dollar Cost Averaging

The following algorithm deposits USD 1000 on every month's start and buys SPY with the deposited cash. It results in a dollar cost averaging investment strategy on the overall market.

```C#
public class PortfolioCashbookAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private decimal _deposit = 1000m;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000);

        // Request SPY data to trade the overall market representative.
        _spy = AddEquity("SPY").Symbol;

        // Deposit each month start and invest in SPY by dollar cost averaging.
        Schedule.On(
            DateRules.MonthStart(_spy),
            TimeRules.AfterMarketOpen(_spy, 0),
            DepositAndRebalance
        );

        // To warm up the price data of SPY to calculate the quantity to be brought.
        SetWarmUp(1);
    }

    private void DepositAndRebalance()
    {
        // Deposit the account currency's preset level ($1000) at the start of the month.
        // Simulate the monthly salary deposit for dollar cost-averaging investment in the market.
        Portfolio.CashBook[AccountCurrency].AddAmount(_deposit);

        // Calculate the number of shares that can be invested using the deposited amount.
        var quantity = Math.Floor(_deposit / Securities[_spy].Price);
        MarketOrder(_spy, quantity);
    }
}
```

Example 2: Crypto Cashbook

This example trades an EMA cross strategy on the ETH-BTC crypto pair in a cash account. To obtain the maximum tradable order size, we need to use the cash book and check the amount of BTC in long ETHBTC trades and in short ETHBTC trades to avoid insufficient initial margin.

C#

```
public class PortfolioCashbookAlgorithm : QCAlgorithm
{
    private Symbol _ethbtc;
    private ExponentialMovingAverage _ema;
    // Set a buffer to avoid insufficient margin due to the price fluctuation.
    private decimal _buffer = 0.01m;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Set the account currency as USDT and the starting cash in a cash account.
        SetBrokerageModel(BrokerageName.Coinbase, AccountType.Cash);
        SetAccountCurrency("USDT", 100);
        SetCash("BTC", 100);
        SetCash("ETH", 2000);

        // We would like to trade the EMA cross between 2 popular cryptos, BTC & ETH,
        // so request ETHBTC data to find trading opportunities.
        _ethbtc = AddCrypto("ETHBTC", Resolution.Minute, market: Market.Coinbase).Symbol;

        // Add automatic-updating EMA indicator for trend trade signal emission.
        _ema = EMA(_ethbtc, 50, Resolution.Daily);
        // Warm up the indicator for its readiness usage immediately.
        WarmUpIndicator(_ethbtc, _ema, Resolution.Daily);
    }

    public override void OnData(Slice slice)
    {
        if (!(slice.Bars.TryGetValue(_ethbtc, out var bar) && _ema.IsReady))
        {
            return;
        }
        decimal quantity;
        // ETHBTC's current price is higher than EMA, suggesting an uptrend.
        if (bar.Close > _ema && !Portfolio[_ethbtc].IsLong)
        {
            // Convert to the corresponding quantity of the quote currency.
            quantity = Portfolio.CashBook["BTC"].Amount * (1m - _buffer) / bar.Close;
        }
        // ETHBTC's current price is below the EMA, suggesting a downtrend.
        else if (bar.Close < _ema && !Portfolio[_ethbtc].IsShort)
        {
            quantity = -Portfolio.CashBook["ETH"].Amount;
        }
        else {
            return;
        }
        // Round down to observe the lot size.
        var lotSize = Securities[_ethbtc].SymbolProperties.LotSize;
        quantity = (long)(quantity / lotSize) * lotSize;
        if (quantity != 0)
        {
            MarketOrder(_ethbtc, quantity);
        }
    }
}
```

Universes

Universes

Key Concepts

Introduction

Universe selection is the process of selecting a basket of assets you may trade. Dynamic universe selection increase diversification and decrease selection bias in your algorithm.

How Universe Selection Works

When you add a universe to your algorithm, LEAN sends a large dataset into a filter function you define. Your filter function needs to return a list of `Symbol` objects. LEAN automatically subscribes to these new symbols and adds them to your algorithm. Your algorithm can do almost anything inside your filter functions, but the goal should be to narrow down the set of securities to the securities that are most applicable for your algorithm.

Selection Functions

The following example filter function selects the 100 most liquid US Equities:

```C#
// Use a fundamental filter function to select the 100 most liquid Equities.
public class MyFundamentalUniverseAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        UniverseSettings.Asynchronous = true;
        AddUniverse(FundamentalFilterFunction);
    }

    private IEnumerable<Symbol> FundamentalFilterFunction(IEnumerable<Fundamental> fundamental)
    {
        return (from c in fundamental
            orderby c.DollarVolume descending
            select c.Symbol).Take(100);
    }
}
```

To learn how to define filter functions for other asset classes, custom data universes , or alternative data universes , see the following pages in this Universes chapter.

Security Changed Events

When your universe adds and removes assets, LEAN notifies your algorithm through the `OnSecuritiesChanged` event handler. The event handler receives a `SecurityChanges` object, which contains references to the added and removed securities. To access the added securities, check the `changes.AddedSecurities` method property. To access the removed securities, check the `changes.RemovedSecurities` method property.

```C#
// Use the OnSecuritiesChanged method to know when an asset is added or removed.
public override void OnSecuritiesChanged(SecurityChanges changes)
{
    // Iterate through the added securities.
    foreach (var security in changes.AddedSecurities)
    {
        Debug($"{Time}: Added {security.Symbol}");
    }
    // Iterate through the removed securities.
    foreach (var security in changes.RemovedSecurities)
    {
        Debug($"{Time}: Removed {security.Symbol}");

        if (security.Invested)
        {
            Liquidate(security.Symbol, tag: "Removed from Universe");
        }
    }
}
```

The preceding example liquidates securities that leave the universe. In this case, LEAN creates a market on open order and frees up buying power when the market opens.

Custom Security Properties

If you need to save data or create objects for each security in the universe, cast the `Security` objects to `dynamic` objects and then save custom members to them . This technique is useful if you want to track stop loss levels or add indicators for each asset in the universe.

```C#
// Cast the security to a dynamic object to add custom properties to it.
public override void OnSecuritiesChanged(SecurityChanges changes)
{
    foreach (var security in changes.AddedSecurities)
    {
        var dynamicSecurity = security as dynamic;

        // Create an SMA indicator with 10 periods for the asset.
        dynamicSecurity.Indicator = SMA(security.Symbol, 10);

        // Warm up the indicator with historical data.
        WarmUpIndicator(security.Symbol, dynamicSecurity.Indicator);
    }
    // Remove the automatic indicator updates when the security leaves the universe.
    foreach (var security in changes.RemovedSecurities)
    {
        DeregisterIndicator((security as dynamic).Indicator);
    }
}
```

Select Current Constituents

If you don't want to make any changes to the current universe, return `Universe.Unchanged` from your filter functions.

```C#
public class MyUniverseAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        UniverseSettings.Asynchronous = true;
        AddUniverse(MyFundamentalFilterFunction);
    }
    // Use Universe.Unchanged to leave the universe unchanged.
    IEnumerable<Symbol> MyFundamentalFilterFunction(IEnumerable<Fundamental> fundamental)
    {
        return Universe.Unchanged;
    }
}
```

Universe Manager

The universe manager tracks all the universes in your algorithm. If you add multiple universe, you can access the constituents of each individual universe. To access the constituents of a universe in a multi-universe algorithm, save references to each universe when you add them.

```C#
private Universe _universe;

// In the Initialize method, add a new universe.
UniverseSettings.Asynchronous = true;
_universe = AddUniverse(MyFundamentalFilterFunction);

// Use the UniverseManager to get information about the assets in each universe.
var universeMembers = UniverseManager[_universe.Configuration.Symbol].Members;
foreach (var kvp in universeMembers)
{
    var symbol = kvp.Key; // The symbol of the asset.
    var security = kvp.Value; // The security in the universe.
}
```

When you remove an asset from a universe, LEAN usually removes the security from the `Members` collection and removes the security subscription. However, it won't remove the security in any of the following situations:

- You own the security.
- You have an open order for the security.
- The security wasn't in the universe long enough to meet the `MinimumTimeInUniverse` setting.

To get only the assets that are currently in the universe, see Selected Securities .

Active Securities

The `QCAlgorithm.ActiveSecurities` is a collection of `Universe.Members` of all universes. The `ActiveSecurities` property of the algorithm class contains all of the assets currently in your algorithm. It is a dictionary where the key is a `Symbol` and the value is a `Security` . When you remove an asset from a universe, LEAN usually removes the security from the `ActiveSecurities` collection and removes the security subscription. However, it won't remove the security in any of the following situations:

- You own the security.
- You have an open order for the security.
- The security wasn't in the universe long enough to meet the `MinimumTimeInUniverse` setting.

When LEAN removes the security, the `Security` object remains in the `Securities` collection for record-keeping purposes, like tracking fees and trading volume.

To improve iteration speed, LEAN removes delisted securities from the `Securities` primary collection. To access all the securities, iterate the `Securities.Total` property.

```
C#

// Access all the securities.
foreach (var security in Securities.Total)
{

}

// Exclude delisted securities.
foreach (var security in Securities.Values)
{

}
```

To get only the assets that are currently in the universe, see Selected Securities .

Selected Securities

The `Selected` property of your `Universe` contains references to all the assets that are currently in the universe. The `Universe.Selected` property differs from the `Universe.Members` property because the `Universe.Members` property can contain more assets than `Universe.Selected` when they have assets with open positions, open order orders, or assets that haven't met the minimum time requirement. To access the `Universe` object, save a reference to the result of the `AddUniverse` method. The following algorithm demonstrates how to use the `Universe.Selected` property to create simple rebalancing strategies:

```
C#

// Rebalance after market open to hold the 5 most liquid Equities.
public class SimpleRebalancingAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        var symbol = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
        // Schedule the universe selection to run at the start of each week.
        var dateRule = DateRules.WeekStart(symbol);
        UniverseSettings.Schedule.On(dateRule);
        // Add a universe of the top 5 most liquid Equities.
        var universe = AddUniverse(Universe.Top(5));
        // 30 minutes after market open, rebalance the portfolio.
        Schedule.On(
            dateRule,
            TimeRules.AfterMarketOpen(symbol, 30),
            () => SetHoldings(
                universe.Selected.Select(symbol => new PortfolioTarget(symbol,
1.0m/universe.Selected.Count)).ToList(),
                liquidateExistingHoldings: true
            )
        );
    }
}
```

Derivative Universes

In a regular universe, you select a basket of assets from the entire universe of securities. In a derivative universe, you select a basket of contracts for an underlying asset. The following derivative universes are available:

- Equity Options
- Futures
- Future Options
- Index Options

Historical Asset Prices

LEAN supports over 100 indicators you can use to generate your trading signals. However, in some cases, it's more efficient to maintain a matrix of historical prices for all the assets in your universe instead of using indicators. For example, say you want to calculate the correlation coefficients between all *n* asset in your universe using a 1-year lookback period. If you were to use the Correlation indicator, when a new asset enters the universe, you would usually need to make a 1-year history request for *n* assets to warm-up the *n* new correlation indicators. These large and frequent history requests can make your algorithm very slow.

The following algorithm demonstrates an alternative, more efficient approach. It defines a `Dictionary` when constructing the algorithm class . When a new asset enters the universe, it adds a new key-value pair to the dictionary. Then, each day, it adds yesterday's open price to each RollingWindow in the dictionary, which covers all the assets in the universe. This approach reduces the size and frequency of history requests you need to make to get the data.

C#

```
using Accord.Statistics;

public class MaintainHistoricalDailyUniversePriceDataAlgorithm : QCAlgorithm
{
    // Create Rolling Window collection to store the historical data.
    private Dictionary<Symbol, RollingWindow<decimal>> _history = new();
    // Define the lookback period.
    private int _lookback = 252;
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);

        // Add a universe of daily data.
        UniverseSettings.Resolution = Resolution.Daily;
        // Filter for any coporate announced a material buyback plan, since they have confidence in their
        // future prospect and the reduction in supply can drive their price up.
        _universe = AddUniverse((fundamentals) => {
            return (from Fundamental f in fundamentals
                   orderby f.MarketCap descending
                   select f.Symbol).Take(10);
        });

        // Create a Scheduled Event to record new daily prices and rebelance the portfolio.
        var spy = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
        Schedule.On(
            DateRules.EveryDay(spy),
            TimeRules.At(0, 1),      // One minute after `OnSecuritiesChanged` runs (in backtests)
            Rebalance
        );
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        // Remove the historical prices of assets that leave the universe.
        foreach (var removed in changes.RemovedSecurities)
        {
            _history.Remove(removed.Symbol);
        }

        // Warm-up the historical data of assets that enter the universe.
        var symbols = changes.AddedSecurities.Select(x => x.Symbol).ToList();
        var history = History<TradeBar>(symbols, _lookback+1, Resolution.Daily);
        foreach (var bars in history.SkipLast(1))
        {
            foreach (var (symbol, bar) in bars)
            {
                if (!_history.TryGetValue(symbol, out var symbolHistory))
                {
                    _history[symbol] = symbolHistory = new(_lookback);
                }

                // If you trade at market open, it might make more sense to generate signals based on daily
                // opening prices.
                symbolHistory.Add(bar.Open);
            }
        }
```

```
        }

        private void Rebalance()
        {
            // Add yesterday's open price to the Dictionary of historical prices.
            var history = History<TradeBar>(_history.Keys, 1, Resolution.Daily).First();
            foreach (var (symbol, bar) in history)
            {
                _history[symbol].Add(bar.Open);
            }

            // To avoid trading errors, select the assets that have a price.
            var symbols = _history.Keys.Where(symbol => Securities[symbol].Price > 0).ToArray();

            // Calculate asset signals for this rebalance.
            // For example, set the signal to give greater weight to uncorrelated assets.
            var priceMatrix = new double[_lookback, symbols.Count()];
            for (int i = 0; i < _lookback; i++)
            {
                for (int j = 0; j < symbols.Count(); j++)
                {
                    priceMatrix[i, j] = (double)_history[symbols[j]][i];
                }
            }
            var correlation = Measures.Correlation(priceMatrix);
            var signals = NormalizedRowAbsoluteSums(correlation);

            // Rebalance the portfolio based on the signals.
            SetHoldings(
                Enumerable
                    .Range(0, signals.Length)
                    .Select(i => new PortfolioTarget(symbols[i], Convert.ToDecimal(signals[i])))
                    .ToList(),
                liquidateExistingHoldings: true
            );
        }

        private double[] NormalizedRowAbsoluteSums(double[,] array)
        {
            var rows = array.GetLength(0);
            var rowAbsSums = new double[rows];

            for (int i = 0; i < rows; i++)
            {
                var rowAbsSum = 0d;
                for (int j = 0; j < array.GetLength(1); j++)
                {
                    rowAbsSum += Math.Abs(array[i, j]);
                }
                rowAbsSums[i] = rowAbsSum;
            }

            var inverseRowSums = rowAbsSums.Select(sum => 1.0d/sum);
            var allSum = inverseRowSums.Sum();
            return inverseRowSums.Select(x => x / allSum).ToArray();
        }
    }
```

Examples

You may refer to the following table for examples of various asset classes:

Asset Class	Universe Type	Example
Equities	Liquidity Universes	Example
Equities	Fundamental Universes	Example
Equities	ETF Constituents Universes	Example
Equities	Chained Universes	Example
Equities	Alternative Data Universes	Example
Equity Options	Equity Option Universes	Example
CFDs	Equity CFD Universes	Example
Futures	Future Universes	Example
Future Options	Future Option Universes	Example
Index Options	Index Option Universes	Example
Cryptos	Crypto Universes	Example
Any	Custom Universes	Example
Any	Dataless Scheduled Universes	Example

Universes

Settings

Introduction

Universe settings and security initializers enable you to configure some properties of the securities in a universe.

Resolution

The `Resolution` setting defines the time period of the asset data. The `Resolution` enumeration has the following members:

To view which resolutions are available for the asset class of your universe, follow these steps:

1. Open the Asset Classes documentation page.
2. Click an asset class.
3. Click Requesting Data .
4. On the Requesting Data page, in the table of contents, click Resolutions .

The default value is `Resolution.Minute` . To change the resolution, in the Initialize method, pass a `resolution` argument to the universe creation method.

```
C#
// Add daily Equity Option data for the SPY.
AddOption("SPY", resolution: Resolution.Daily);
```

Leverage

The `Leverage` setting is a `decimal` that defines the maximum amount of leverage you can use for a single asset in a non-derivative universe. The default value is `Security.NullLeverage` (0). To change the leverage, in the Initialize method, adjust the algorithm's `UniverseSettings` before you add the universe.

```
C#
// Assign 2x leverage for all securities in the universe.
UniverseSettings.Leverage = 2.0m;
// Add a universe of the 50 most liquid US Equities.
AddUniverse(Universe.Top(50));
```

Fill Forward

The `FillForward` setting is a `bool` that defines whether or not too fill forward data. The default value is `true` . To disable fill forward in non-derivative universes, in the Initialize method, adjust the algorithm's `UniverseSettings` before you add the universe.

```
C#
// Disable fill-forward data before creating the universe (it's enabled by default).
UniverseSettings.FillForward = false;
// Add the 50 Equities to the universe that have the most dollar trading volume.
AddUniverse(Universe.Top(50));
```

To disable fill forward in derivative universes, pass a `fillForward` argument to the universe creation method.

```
// Disable fill-forward in the VIX Index Option universe.
AddIndexOption("VIX", fillForward: false);
```

Extended Market Hours

The `ExtendedMarketHours` setting is a `bool` that defines the trading schedule. If it's `true` , your algorithm receives price data for all trading hours. If it's `false` , your algorithm receives price data only for regular trading hours. The default value is `false` .

You only receive extended market hours data if you create the subscription with an intraday resolution. If you create the subscription with daily resolution, the daily bars only reflect the regular trading hours.

To view the trading schedule of an asset, open Asset Classes , click an asset class, then click Market Hours .

To enable extended market hours in non-derivative universes, in the Initialize method, adjust the algorithm's `UniverseSettings` before you add the universe.

```
// Enable pre- and post-market hours data for assets in the universe.
UniverseSettings.ExtendedMarketHours = true;
// Add the 50 Equities to the universe that have the most dollar trading volume.
AddUniverse(Universe.Top(50));
```

To enable extended market hours in derivative universes, pass an `extendedMarketHours` argument to the universe creation method.

```
// Request pre- and post-market hours data for a Futures universe.
AddFuture(Futures.Currencies.BTC, extendedMarketHours: true);
```

Minimum Time in Universe

The `MinimumTimeInUniverse` setting is a `TimeSpan` object that defines the minimum amount of time an asset must be in the universe before the universe can remove it. The default value is `TimeSpan.FromDays(1)` . To change the minimum time, in the Initialize method, adjust the algorithm's `UniverseSettings` before you add the universe.

```
// Keep each security in the universe for a minimum of 7 days.
UniverseSettings.MinimumTimeInUniverse = TimeSpan.FromDays(7);
// Add the top 50 most liquid Equities to the universe.
AddUniverse(Universe.Top(50));
```

Data Normalization Mode

The `DataNormalizationMode` setting is an enumeration that defines how historical data is adjusted. This setting is only applicable for US Equities and Futures.

US Equities

In the case of US Equities, the data normalization mode affects how historical data is adjusted for corporate actions . To view all the available options, see Data Normalization . The default value is `DataNormalizationMode.Adjusted` . To change the data normalization mode, in the Initialize method, adjust the algorithm's `UniverseSettings` before you add the universe.

```
// Pipe raw price data (unadjusted for splits and dividends) into the algorithm.
// Equity Options require raw data and it improves the accuracy of modeling historical periods.
UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;
// Add a universe of the 50 most liquid US Equities.
AddUniverse(Universe.Top(50));
```

Futures

In the case of Futures, the data normalization mode affects how historical data of two contracts is stitched together to form the continuous contract . To view all the available options, see Data Normalization . The default value is `DataNormalizationMode.Adjusted` . To change the data normalization mode, in the Initialize method, pass a `dataNormalizationMode` argument to the `AddFuture` method.

```
// Set a specific data normalization mode for the continuous contract
// with the dataNormalizationMode parameter of the AddFuture method.
AddFuture(Futures.Currencies.BTC, dataNormalizationMode: DataNormalizationMode.BackwardsRatio);
```

Contract Depth Offset

The `ContractDepthOffset` setting is an `int` that defines which contract to use for the continuous Futures contract. 0 is the front month contract, 1 is the following back month contract, and 3 is the second back month contract. The default value is 0. To change the contract depth offset, in the Initialize method, pass a `contractDepthOffset` argument to the `AddFuture` method.

```
// To adjust the contract depth offset to the second back month contract,
// pass the value 3 to the contractDepthOffset argument in the AddFuture method.
AddFuture(Futures.Currencies.BTC, contractDepthOffset: 3);
```

Asynchronous Selection

The `Asynchronous` setting is a `bool` that defines whether or not LEAN can run universe selection asynchronously, utilizing concurrent execution to increase the speed of your algorithm. The default value for this setting is `false` . If you enable this setting, abide by the following rules:

- Don't make any history requests in your filter function . History requests can only provide data up to the algorithm time , but if your filter function runs asynchronously, the algorithm time may not be what you expect.
- Don't use the portfolio, security, or orders state. The `Portfolio` , `Securities` , and `Transactions` objects are also functions of the algorithm time.
- If your filter function updates a class variable, don't update the class variable anywhere else in your algorithm.

To enable asynchronous universe selection, in the Initialize method, adjust the algorithm's `UniverseSettings` before you add the universe.

```
// To speed up algorithm performance, enable asynchronous universe selection.
// Only use this when the code is self-contained and stateless.
UniverseSettings.Asynchronous = true;
AddUniverse(Universe.Top(50));
```

Schedule

The `Schedule` setting defines the selection schedule of the universe. Most universes run on a daily schedule. To change the

selection schedule, call the `UniverseSettings.Schedule.On` method with an `IDateRule` object before you add the universe.

```
C#
// Run universe selection on a monthly basis.
UniverseSettings.Schedule.On(DateRules.MonthStart());
AddUniverse(Universe.Top(50));
```

The following table describes the supported `DateRules` :

Member	Description
`DateRules.SetDefaultTimeZone(DateTimeZone timeZone);`	Sets the time zone for the `DateRules` object used in all methods in this table. The default time zone is the algorithm time zone .
`DateRules.On(int year, int month, int day)`	Trigger an event on a specific date.
`DateRules.On(params DateTime[] dates)`	Trigger an event on specific dates.
`DateRules.EveryDay()`	Trigger an event all seven days of the week.
`DateRules.EveryDay(Symbol symbol, bool extendedMarketHours = false)`	Trigger an event every day a specific symbol is trading.
`DateRules.Every(params DayOfWeek[] days)`	Trigger an event on specific days throughout the week. To view the `DayOfWeek` enum members, see DayOfWeek Enum in the .NET documentation.
`DateRules.MonthStart(int daysOffset = 0)`	Trigger an event on the first day of each month plus an offset.
`DateRules.MonthStart(Symbol symbol, int daysOffset = 0)`	Trigger an event on the first tradable date of each month for a specific symbol plus an offset.
`DateRules.MonthEnd(int daysOffset = 0)`	Trigger an event on the last day of each month minus an offset.
`DateRules.MonthEnd(Symbol symbol, int daysOffset = 0)`	Trigger an event on the last tradable date of each month for a specific symbol minus an offset.
`DateRules.WeekStart(int daysOffset = 0)`	Trigger an event on the first day of each week plus an offset.
`DateRules.WeekStart(Symbol symbol, int daysOffset = 0)`	Trigger an event on the first tradable date of each week for a specific symbol plus an offset.
`DateRules.WeekEnd(int daysOffset = 0)`	Trigger an event on the last day of each week minus an offset.
`DateRules.WeekEnd(Symbol symbol, int daysOffset = 0)`	Trigger an event on the last tradable date of each week for a specific symbol minus an offset.
`DateRules.YearStart(int daysOffset = 0)`	Trigger an event on the first day of each year plus an offset.
`DateRules.YearStart(Symbol symbol, int daysOffset = 0)`	Trigger an event on the first tradable date of each year for a specific symbol plus an offset.

`DateRules.YearEnd(int daysOffset = 0)`	Trigger an event on the last day of each year minus an offset.
`DateRules.YearEnd(Symbol symbol, int daysOffset = 0)`	Trigger an event on the last tradable date of each year for a specific symbol minus an offset.
`DateRules.Today`	Trigger an event once today.
`DateRules.Tomorrow`	Trigger an event once tomorrow.

To define custom date rules, create a `FuncDateRule` object. The `FuncDateRule` constructor expects a `name` argument of type `string` and a `getDatesFunction` argument of type `Func<DateTime, DateTime, IEnumerable<DateTime>>` . The `getDatesFunction` function receives the start and end dates of the algorithm and returns a list of dates for the date rule. In live trading, the end date is 12/31/2025. The following example demonstrates how to define a date rule that represents the 10th day of each month:

```
C#

// Create a date rule that specifies the 10th day of each month.
var dateRule = new FuncDateRule(
    name: "10th_day_of_the_month",
    getDatesFunction: (start, end) => Enumerable.Range(start.Year, end.Year - start.Year + 1)
        .SelectMany(year => Enumerable.Range(1, 12).Select(month => new DateTime(year, month, 10)))
);
```

Some date rules require a Symbol. However, they accept a string representation of the Symbol as well. If your algorithm doesn't add the security , it assumes the string refers to a US Equity Symbol. For example, if you use the `DateRules.EveryDay("ES")` method, it assumes that 'ES' refers to a US Equity. To create a Symbol without adding a security to your algorithm, call the `QuantConnect.Symbol.Create` method.

```
C#

var symbol = QuantConnect.Symbol.Create("ES", SecurityType.Future, Market.CME);
```

In live trading, scheduled universes run at roughly 8 AM Eastern Time to ensure there is enough time for the data to process.

Configure Universe Securities

Instead of configuring global universe settings, you can individually configure the settings of each security in the universe with a security initializer. Security initializers let you apply any security-level reality model or special data requests on a per-security basis. To set the security initializer, in the `Initialize` method, call the `SetSecurityInitializer` method and then define the security initializer.

```
C#

// A custom security initializer can override default models such as
// setting new fee and fill models for the security.
SetSecurityInitializer(CustomSecurityInitializer);

private void CustomSecurityInitializer(Security security)
{
    security.SetFeeModel(new ConstantFeeModel(0, "USD"));
}
```

For simple requests, you can use the functional implementation of the security initializer. This style lets you configure the security object with one line of code.

```
// Disable the trading fees for each security by passing a functional
// implementation for the SetSecurityInitializer argument.
SetSecurityInitializer(security => security.SetFeeModel(new ConstantFeeModel(0, "USD")));
```

If you call the `SetSecurityInitializer` method, it overwrites the default security initializer. The default security initializer uses the security-level reality models of the brokerage model to set the following reality models of each security:

- Fill
- Slippage
- Fee
- Buying Power
- Settlement
- Short availability
- Margin Interest Rate

The default security initializer also sets the leverage of each security and intializes each security with a seeder function. To extend upon the default security initializer instead of overwriting it, call `AddSecurityInitializer` before you create the subscriptions.

```
public class AddSecurityInitializerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the security initializer to set models of assets.
        AddSecurityInitializer(CustomSecurityInitializer);
    }

    private void CustomSecurityInitializer(Security security)
    {
        // Overwrite some of the reality models
        security.SetFeeModel(new ConstantFeeModel(0, "USD"));
    }
}
```

Examples

The following examples demonstrate some common practices for universe settings.

Example 1: Weekly-Updating Liquid Universe

The following algorithm demonstrates daily EMA cross, trading on the most liquid stocks. The universe is set to be updated weekly. Various universe settings have been set to simulate the brokerage environment best and for trading needs.

C#

```
public class UniverseSettingsAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // To avoid over-updating the universe, which causes insufficient time to earn the trend, update the universe
weekly.
        UniverseSettings.Schedule.On(DateRules.WeekStart());
        UniverseSettings.MinimumTimeInUniverse = TimeSpan.FromDays(7);
        // Rebalancing frequency is daily, so we only need to subscribe to daily resolution.
        UniverseSettings.Resolution = Resolution.Daily;
        // To best simulate IB's margin requirement set the leverage to 50%.
        UniverseSettings.Leverage = 2;
        // Since we trade by market order, we cannot use extended market hours data to fill.
        UniverseSettings.ExtendedMarketHours = false;
        // We want to trade the EMA with raw price but not altered by splits.
        UniverseSettings.DataNormalizationMode = DataNormalizationMode.SplitAdjusted;

                // Only trade on the top 10 most traded stocks since they have the most popularity to drive trends.
        AddUniverse(Universe.Top(10));
            }

    public override void OnData(Slice slice)
    {
        foreach (var (symbol, bar) in slice.Bars)
        {
            // Trade the trend by EMA crossing.
            var ema = (Securities[symbol] as dynamic).ema;
            if (bar.Close > ema)
            {
                SetHoldings(symbol, 0.05m);
            }
            else
            {
                SetHoldings(symbol, -0.05m);
            }
        }
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach (var removed in changes.RemovedSecurities)
        {
            // Liquidate the ones leaving the universe since we will not trade their EMA anymore.
            Liquidate(removed.Symbol);
            // Deregister the EMA indicator to free resources
            DeregisterIndicator((removed as dynamic).ema as ExponentialMovingAverage);
        }

        foreach (var added in changes.AddedSecurities)
        {
            var security = added as dynamic;
            // Create EMA indicator for trend trading signals.
            security.ema = EMA(added.Symbol, 50, Resolution.Daily);
            // Warm up the EMA indicator to ensure its readiness for immediate use.
            WarmUpIndicator(added.Symbol, (ExponentialMovingAverage)security.ema, Resolution.Daily);
        }
    }
}
```

Universes

Equity

You can create any of the following dynamic universes of equity. Click one to learn more.

Liquidity Universes

Fundamental Universes

ETF Constituents Universes

Chained Universes

Legacy Fundamental Universes

Alternative Data Universes

See Also

US Equity

Equity

Liquidity Universes

Introduction

A liquidity universe lets you select a set of stocks with the greatest dollar volume in the market.

Create Universes

To add a dollar volume universe, call the `Universe.DollarVolume.Top` helper method and pass the result to the `AddUniverse` method.

```
C#
// Add the 50 stocks with the highest dollar volume.
UniverseSettings.Asynchronous = true;
AddUniverse(Universe.Top(50));
```

Selection Frequency

Equity universes run on a daily basis by default. To adjust the selection schedule, see Schedule .

Examples

The following examples demonstrate some common liquidity universes for US Equities.

Example 1: Minute-Resolution Universe

The following algorithm asynchronously selects the 10 most liquid US Equities as the universe. It uses minute resolution data to provide the most realistic results.

```
C#
public class MinuteLiquidUniverseAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Use asynchronous universe selection to speed up your algorithm.
        UniverseSettings.Asynchronous = true;
        // Use minute resolution data so orders fill at realistic prices.
        UniverseSettings.Resolution = Resolution.Minute;
        // Add a universe of the 10 most liquid US Equities.
        AddUniverse(Universe.Top(10));
    }
}
```

Example 2: Trade Liquid Assets off Bollinger Bands

The following algorithm asynchronously selects the 500 most liquid US Equities as the universe. Each day, it forms a portfolio of the assets in the universe that are below their lower Bollinger Band .

C#

```
public class AsynchronousLiquidUniverseAlgorithm : QCAlgorithm
{
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        Settings.AutomaticIndicatorWarmUp = true;
        // When a new asset enters the universe, seed its current price so you can trade it right away.
        Settings.SeedInitialPrices = true;
        // Use asynchronous universe selection to speed up the algorithm.
        UniverseSettings.Asynchronous = true;
        // Use daily data. It doesn't affect accuracy since the trades fill at the daily market open.
        UniverseSettings.Resolution = Resolution.Daily;
        // Add a universe of the 500 most liquid Equities.
        _universe = AddUniverse(Universe.Top(500));
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        // Create Bollinger Band indicators for the universe constituents.
        foreach (var security in changes.AddedSecurities)
        {
            (security as dynamic).BB = BB(security.Symbol, 60, 2);
        }
        // Deregister the indicators when assets leave the universe.
        foreach (var security in changes.RemovedSecurities)
        {
            DeregisterIndicator((security as dynamic).BB);
        }
    }

    public override void OnData(Slice slice)
    {
        // Ensure there are TradeBar objects in the current slice.
        if (slice.Bars.Count == 0)
        {
            return;
        }
        // Select the Equities that are below their lower Bollinger Band.
        var selected = _universe.Selected
            .Select(symbol => (Securities[symbol] as dynamic))
            .Where(security => security.BB.IsReady && security.Price != 0 && security.Price <
security.BB.LowerBand.Current.Value);
        // Form an equal-weighted portfolio with the selected assets.
        var weight = 1m / selected.Count();
        SetHoldings(selected.Select(security => new PortfolioTarget(security.Symbol, weight)).ToList(),
liquidateExistingHoldings: true);
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
UniverseSelectionDefinitionsAlgorithm.cs C#

Equity

Fundamental Universes

Introduction

A fundamental universe lets you select stocks based on corporate fundamental data. This data is powered by Morningstar® and includes approximately 8,100 tickers (including delisted companies and excluding ETFs, ADRs, and OTC) with 900 properties each.

Create Universes

To add a fundamental universe, in the `Initialize` method, pass a filter function to the `AddUniverse` method. The filter function receives a list of `Fundamental` objects and must return a list of `Symbol` objects. The `Symbol` objects you return from the function are the constituents of the fundamental universe and LEAN automatically creates subscriptions for them. Don't call `AddEquity` in the filter function.

```
C#
public class MyUniverseAlgorithm : QCAlgorithm {
    private Universe _universe;
    public override void Initialize()
    {
        UniverseSettings.Asynchronous = true;
        // Add a fundamental universe with a custom filter function.
        _universe = AddUniverse(FundamentalFilterFunction);
    }

    private IEnumerable<Symbol> FundamentalFilterFunction(IEnumerable<Fundamental> fundamental)
    {
         // Select US Equities that have fundamental data.
         return (from f in fundamental
                where f.HasFundamentalData
                select f.Symbol);
    }
}
```

`Fundamental` objects have the following attributes:

Example

The simplest example of accessing the fundamental object would be harnessing the iconic PE ratio for a stock. This is a ratio of the price it commands to the earnings of a stock. The lower the PE ratio for a stock, the more affordable it appears.

```
C#
// Select the top 50 most liquid US Equities and then select the 10 with
// lowest PE ratios.
UniverseSettings.Asynchronous = true;
_universe = AddUniverse(
    fundamental => (from f in fundamental
        where f.Price > 10 && f.HasFundamentalData && !Double.IsNaN(f.ValuationRatios.PERatio)
        orderby f.DollarVolume descending
        select f).Take(100)
        .OrderBy(f => f.ValuationRatios.PERatio).Take(10)
        .Select(f => f.Symbol));
```

Asset Categories

In addition to valuation ratios, the US Fundamental Data from Morningstar has many other data point attributes, including over

200 different categorization fields for each US stock. Morningstar groups these fields into sectors, industry groups, and industries.

Sectors are large super categories of data. To get the sector of a stock, use the `MorningstarSectorCode` property.

```
C#

// Select the US Equities in the technology sector.
var tech = fundamental.Where(x => x.AssetClassification.MorningstarSectorCode == MorningstarSectorCode.Technology);
```

Industry groups are clusters of related industries that tie together. To get the industry group of a stock, use the `MorningstarIndustryGroupCode` property.

```
C#

// Select the US Equities in the agricluture industry group.
var ag = fundamental.Where(x => x.AssetClassification.MorningstarIndustryGroupCode ==
MorningstarIndustryGroupCode.Agriculture);
```

Industries are the finest level of classification available. They are the individual industries according to the Morningstar classification system. To get the industry of a stock, use the `MorningstarIndustryCode` property.

```
C#

// Select the US Equities in the coal industry.
var coal = fundamental.Where(x => x.AssetClassification.MorningstarIndustryCode == MorningstarIndustryCode.Coal);
```

Practical Limitations

Fundamental universes allow you to select an unlimited universe of assets to analyze. Each asset in the universe consumes approximately 5MB of RAM, so you may quickly run out of memory if your universe filter selects many assets. If you backtest your algorithms in the Algorithm Lab, familiarize yourself with the RAM capacity of your backtesting and live trading nodes . To keep your algorithm fast and efficient, only subscribe to the assets you need.

Data Availability

`Fundamental` objects can have NaN values for some of their properties. Before you sort the `Fundamental` objects by one of the properties, filter out the objects that have a NaN value for the property.

```
C#

private IEnumerable<Symbol> FundamentalFilterFunction(IEnumerable<Fundamental> fundamentals)
{
    return fundamentals
        // Select objects that have a value for the fundamental property.
        .Where(f => f.HasFundamentalData && !Double.IsNaN(f.ValuationRatios.PERatio))
        // Sort the objects by the fundamental property.
        .OrderBy(f => f.ValuationRatios.PERatio)
        .Take(10)
        .Select(x => x.Symbol);
}
```

Direct Access

To get fundamental data for Equities in your algorithm, use the `Fundamentals` property of the `Equity` objects. The fundamental data represent the company's fundamentals for the current algorithm time.

```
C#

var fundamentals = Securities[_symbol].Fundamentals;
```

To get fundamental data for Equities, regardless of whether or not you have subscribed to them in your algorithm, call the `Fundamentals` method. If you pass one `Symbol` , the method returns a `Fundamental` object. If you pass a list of `Symbol` objects, the method returns a list of `Fundamental` objects. The fundamental data represents the corporate fundamentals for the current algorithm time.

```
C#

// Single asset
var ibm = QuantConnect.Symbol.Create("IBM", SecurityType.Equity, Market.USA);
var ibmFundamental = Fundamentals(ibm);

// Multiple assets
var nb = QuantConnect.Symbol.Create("NB", SecurityType.Equity, Market.USA);
var fundamentals = Fundamentals(new List<Symbol>{ nb, ibm }).ToList();
```

Data Availability

Some assets don't have fundamentals (for example, ETFs) and the Morningstar dataset doesn't provide fundamentals for all US Equities. To check if fundamental data is available for an asset, use the `HasFundamentalData` property.

```
C#

var hasFundamentalData = Securities[_symbol].Fundamentals.HasFundamentalData;
```

Object References

If you save a reference to the `Fundamentals` object or its properties, you can access the fundamental properties as they change over time.

```
C#

_fundamentals = Securities[_symbol].Fundamentals;
var earningRatios = _fundamentals.EarningRatios;
```

Historical Data

To get historical fundamental data, call the `History` method. The return type depends on how you call the method.

```
C#

var ibm = QuantConnect.Symbol.Create("IBM", SecurityType.Equity, Market.USA);

// Fundamental objects
var fundamentalHistory = History<Fundamental>(ibm, TimeSpan.FromDays(30));

// Fundamentals objects for all US Equities (including delisted companies)
var fundamentalsHistory = History<Fundamentals>(TimeSpan.FromDays(30));

// Collection of Fundamental objects for all US Equities (including delisted companies)
var collectionHistory = History(_universe, 30, Resolution.Daily);
foreach (var fundamental in collectionHistory)
{
    // Cast to Fundamental is required
    var highestMarketCap = fundamental.OfType<Fundamental>().OrderByDescending(x => x.MarketCap).Take(5);
}
```

For more information about historical fundamental data, see Equity Fundamental Data .

Selection Frequency

Equity universes run on a daily basis by default. To adjust the selection schedule, see Schedule .

Live Trading Considerations

The live data for fundamental universe selection arrives at 6/7 AM Eastern Time (ET), so fundamental universe selection runs for live algorithms between 7 and 8 AM ET. This timing allows you to place trades before the market opens. Don't schedule anything for midnight because the universe selection data isn't ready yet.

Examples

The following examples demonstrate some common practices for fundamental universes.

Example 1: 500 Stocks >10/*Shareand* > 10M in Daily Trading Volume

The following algorithm selects the 500 most liquid US Equities above 10/*shareand*10M in daily volume.

```C#
public class LiquidNonPennyStocksUniverseAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Configure the universe to update at the start of each month. Most of the top 500 doesn't change very
        // frequently.
        UniverseSettings.Schedule.On(DateRules.MonthStart());
        // Add a universe with custom selection rules for filtering.
        AddUniverse(fundamental => (from f in fundamental
            where f.Price > 10 && f.DollarVolume > 10000000
            orderby f.DollarVolume descending
            select f.Symbol).Take(500));
    }
}
```

Example 2: 10 Stocks Above Their 200-Day EMA With >$1B of Daily Trading Volume

Another common request is to filter the universe by a technical indicator, such as only picking stocks above their 200-day Exponential Moving Average (EMA). The `Fundamental` object has adjusted price and volume information, so you can do any price-related analysis. The following algorithm defines a separate class to contain the indicator of each asset.

```C#
using System.Collections.Concurrent;

public class UpTrendLiquidUniverseAlgorithm : QCAlgorithm
{
    // Create a concurrent dictionary to store the EMA data for universe selection.
    private ConcurrentDictionary<Symbol, SelectionData> _selectionDataBySymbol = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Add the custom universe.
        AddUniverse(SelectAssets);
    }

    private IEnumerable<Symbol> SelectAssets(IEnumerable<Fundamental> fundamental)
    {
        return (from f in fundamental
            // Create/Update the EMA indicators of each stock.
            let avg = _selectionDataBySymbol.GetOrAdd(f.Symbol, sym => new SelectionData(200))
            where avg.Update(f.EndTime, f.AdjustedPrice)
            // Select the Equities that are above their EMA and have a daily volume of $1B.
            // These assets are in an uptrend and are very liquid.
            where avg.Ema.IsReady && f.Price > avg.Ema.Current.Value && f.DollarVolume > 1000000000
            // Select the 10 most liquid Equities to avoid extra slippage.
            orderby f.DollarVolume descending
            select f.Symbol).Take(10);
    }
}

// Create a separate class to contain the EMA information of each asset.
class SelectionData
{
    public readonly ExponentialMovingAverage Ema;

    // Create an EMA indicator for trend estimation and filtering.
    public SelectionData(int period)
    {
        Ema = new ExponentialMovingAverage(period);
    }

    // Update your variables and indicators with the latest data.
    // You may also want to use the History API here to warm-up the indicator.
    public bool Update(DateTime time, decimal value)
    {
        return Ema.Update(time, value);
    }
}
```

In this example, the `SelectionData` class group variables for the universe selection and updates the indicator of each asset. We highly recommend you follow this pattern to keep your algorithm tidy and bug free. The following snippet shows an example implementation of the `SelectionData` class, but you can make this whatever you need to store your custom universe filters. Note that the preceding `SelectionData` class uses a manual EMA indicator instead of the automatic version . For more information about universes that select assets based on indicators, see Indicator Universes . You need to use a `SelectionData` class instead of assigning the EMA to the `Fundamental` object because you can't create custom properties on `Fundamental` objects like you can with `Security` objects.

Example 3: 10 Stocks Furthest Above their 10-day SMA of Volume

The process to get the 10-day Simple Moving Average (SMA) stock volume is the same process as in Example 2. First, define a `SelectionData` class that performs the averaging. This class tracks the ratio of today's volume relative to historical volumes. You can use this ratio to select assets that are above their 10-day SMA and sort the results by the Equities that have had the biggest jump since yesterday.

```C#
using System.Collections.Concurrent;

public class HighRelativeVolumeUniverseAlgorithm : QCAlgorithm
{
    // Create a dictionary to store the EMA data for universe selection.
    private ConcurrentDictionary<Symbol, SelectionData> _selectionDataBySymbol = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Add a universe with custom selection rules for filtering.
        AddUniverse(SelectAssets);
    }

    private IEnumerable<Symbol> SelectAssets(IEnumerable<Fundamental> fundamental)
    {
        return (from f in fundamental
            // Create/Update the volume SMA indicator of each stock.
            let avg = _selectionDataBySymbol.GetOrAdd(f.Symbol, sym => new SelectionData(f.Symbol, 10))
            where avg.Update(f.EndTime, f.Volume)
            // Select the Equities with higher trading volume than their SMA, indicating higher capital flow.
            where avg.VolumeRatio > 1
            // Select the 10 Equities with the highest relative volume, since they have the highest capactity
            // for scalp-trading or intra-day movement.
            orderby avg.VolumeRatio descending
            select f.Symbol).Take(10);
    }
}

// Define a separate class to contain and calculate the SMA of each Equity.
class SelectionData
{
    public readonly Symbol Symbol;
    public readonly SimpleMovingAverage VolumeSma;
    public decimal VolumeRatio;

    public SelectionData(Symbol symbol, int period)
    {
        // Create an SMA of volume to track the popularity of the stock.
        Symbol = symbol;
        VolumeSma = new SimpleMovingAverage(period);
    }

    public bool Update(DateTime time, decimal value)
    {
        // Update the SMA with today's data and calculate the relative volume position for filtering.
        var ready = VolumeSma.Update(time, value);
        VolumeRatio = VolumeSma.Current.Value != 0m ? value / VolumeSma.Current.Value : -1m;
        return ready;
    }
}
```

Example 4: 10 "Fastest Moving" Stocks With a 50-Day EMA > 200 Day EMA

You can construct complex universe filters with the `SelectionData` helper class pattern. To view a full example of this algorithm, see the EmaCrossUniverseSelectionAlgorithm in the LEAN GitHub repository or take the related Boot Camp lesson .

Example 5: Piotroski F-Score

To view this example, see the Piotroski F-Score Investing Research post.

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
FundamentalUniverseSelectionAlgorithm.cs C#

Equity

ETF Constituents Universes

Introduction

An ETF constituents universe lets you select a universe of securities in an exchange traded fund. The US ETF Constituents dataset includes 2,650 US ETFs you can use to create your universe.

Create Universes

To add an ETF Constituents universe, call the `Universe.ETF` method.

```C#
public class ETFConstituentsAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        UniverseSettings.Asynchronous = true;
        AddUniverse(Universe.ETF("SPY"));
    }
}
```

The following table describes the `ETF` method arguments:

Argument: `etfTicker` The ETF ticker. To view the supported ETFs in the US ETF Constituents dataset, see Supported ETFs . Data Type: `string` | Default Value: `None`
Argument: `universeSettings` The universe settings . If you don't provide an argument, it uses the algorithm `UniverseSettings` . Data Type: `UniverseSettings` | Default Value: `null`
Argument: `universeFilterFunc` A function to select some of the ETF constituents for the universe. If you don't provide an argument, it selects all of the constituents. Data Type: `Func<IEnumerable<ETFConstituentUniverse>, IEnumerable<Symbol>>` | Default Value: `null`

To select a subset of the ETF constituents, provide a `universeFilterFunc` argument. The filter function receives `ETFConstituentUniverse` objects, which represent one of the ETF constituents. `ETFConstituentUniverse` objects have the following attributes:

```C#
public class ETFConstituentsAlgorithm : QCAlgorithm
{
    private Universe _universe;
    public override void Initialize()
    {
        UniverseSettings.Asynchronous = true;
        _universe = Universe.ETF("SPY", UniverseSettings, ETFConstituentsFilter);
        AddUniverse(_universe);
    }

    private IEnumerable<Symbol> ETFConstituentsFilter(IEnumerable<ETFConstituentUniverse> constituents)
    {
        // Get the 10 securities with the largest weight in the index
        return constituents.OrderByDescending(c => c.Weight).Take(10).Select(c => c.Symbol);
    }
}
```

Historical Data

To get historical ETF constituents data, call the `History` method with the `Universe` object and the lookback period. The return type is a `IEnumerable<BaseDataCollection>` and you have to cast its items to `ETFConstituentUniverse` .

```C#
var history = History(_universe, 30, Resolution.Daily);
foreach (var constituents in history)
{
    foreach (ETFConstituentUniverse constituent in constituents)
    {
        Log($"{constituent.Symbol} weight at {constituent.EndTime}: {constituent.Weight}");
    }
}
```

For more information about ETF Constituents data, see US ETF Constituents .

Selection Frequency

Equity universes run on a daily basis by default. To adjust the selection schedule, see Schedule .

Examples

The following examples demonstrate some common practices for ETF constituent universes.

Example 1: Short the Smallest ETF Constituents

A subset of the SPY constituents outperform the SPY while many of the constituents underperform the overall index. In an attempt to exclude the ETF constituents that underperform the index, the following algorithm buys the SPY and shorts the 100 assets in the index with the smallest weight in the ETF.

```C#
public class ETFConstituentsUniverseAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private Dictionary<Symbol, decimal> _etfWeightBySymbol;
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(10000000);
        Settings.MinimumOrderMarginPortfolioPercentage = 0m;
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // To avoid over-trading and high transaction costs, refilter and rebalance weekly.
        UniverseSettings.Schedule.On(DateRules.WeekEnd());
        // Add the SPY ETF.
        _spy = AddEquity("SPY").Symbol;
        // Add a universe of the SPY constituents.
        _universe = AddUniverse(Universe.ETF(_spy, universeFilterFunc: SelectAssets));
        // Create a Scheduled Event to rebalance the portfolio.
        Schedule.On(DateRules.WeekStart(), TimeRules.At(9, 0), Rebalance);
    }

    private IEnumerable<Symbol> SelectAssets(IEnumerable<ETFConstituentUniverse> constituents)
    {
        // Cache the constituent weights in a dictionary for filtering and position sizing.
        _etfWeightBySymbol = constituents
            .Where(c => c.Weight.HasValue)
            .ToDictionary(c => c.Symbol, c => c.Weight.Value);
        // Select the 100 consituents with the smallest weight in the ETF.
        // They should have negative excess return.
        return _etfWeightBySymbol
            .OrderBy(x => x.Value)
            .Take(100)
            .Select(x => x.Key);
    }

    private void Rebalance()
    {
        // Get the ETF weight of all the assets currently in the universe.
        var weightBySymbol = _universe.Selected
            // To avoid trading errors, skip assets that have no price yet.
            .Where(symbol => Securities[symbol].Price != 0)
            .Select(symbol => new { Symbol = symbol, Weight = _etfWeightBySymbol[symbol] })
            .ToDictionary(x => x.Symbol, x => x.Weight);
        // Buy the SPY to eliminate systematic risk.
        var targets = new List<PortfolioTarget>
        {
            new PortfolioTarget(_spy, 1.0m-weightBySymbol.Sum(kvp => kvp.Value))
        };
        // Sell the 100 ETF constituents with the lowest weight due to expected negative excess return.
        targets.AddRange(weightBySymbol.Select(x => new PortfolioTarget(x.Key, -x.Value)));
        // Place orders to rebalance the portfolio.
        SetHoldings(targets, liquidateExistingHoldings: true);
    }
}
```

Example 2: SPY Constituents in an Uptrend

The following example chains an ETF constituents universe and a fundamental universe . It first selects all the constituents of the SPY ETF and then filters them down in the fundamental universe filter to select the assets that are trading above their average price over the last 200 days. The output of the fundamental universe selection method is the output of the chained universe. The `Fundamental` objects that the fundamental universe filter function recieves contains the prices of the ETF constituents. By chaining the ETF constituents universe into the fundamental universe, you can update the indicators with the price instead of making a history request .

```C#
public class ChainedUniverseAlgorithm : QCAlgorithm
{
    private Dictionary<Symbol, SelectionData> _selectionDataBySymbol = new();
    private Universe _universe;
```

```
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Seed the price of each asset that enters the universe with its last known price so you can trade it
        // on the same morning it enters the universe without getting warnings.
        Settings.SeedInitialPrices = true;
        // Add a chained universe that selects the SPY constituents trading above their 200-day SMA.
        var spy = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
        _universe = AddUniverse(Universe.ETF(spy), FundamentalSelection);
        // Trade daily at market open since the trading signal is generated on a daily resolution.
        Schedule.On(DateRules.EveryDay(spy), TimeRules.AfterMarketOpen(spy, 1), Rebalance);
    }

    public IEnumerable<Symbol> FundamentalSelection(IEnumerable<Fundamental> fundamental)
    {
        // Create/Update an SMA indicator for each asset that enters the ETF.
        var universeSymbols = new List<Symbol>();
        foreach (var f in fundamental)
        {
            universeSymbols.Add(f.Symbol);
            if (!_selectionDataBySymbol.ContainsKey(f.Symbol))
            {
                _selectionDataBySymbol[f.Symbol] = new SelectionData(this, f.Symbol);
            }
            _selectionDataBySymbol[f.Symbol].Update(Time, f.AdjustedPrice);
        }

        // Remove indicators for assets that are no longer in the ETF to release the computational resources.
        var symbolsToRemove = _selectionDataBySymbol.Keys.Where(symbol =>
!universeSymbols.Contains(symbol)).ToList();
        foreach (var symbol in symbolsToRemove)
        {
            _selectionDataBySymbol.Remove(symbol);
        }

        // Select the Equities trading above their SMA as we estimate them to be in uptrend and still rising.
        var selected = _selectionDataBySymbol.Where(kvp => kvp.Value.IsAboveSma)
            .Select(kvp => kvp.Key)
            .ToList();

        // Plot the results.
        Plot("Universe", "Possible", fundamental.Count());
        Plot("Universe", "Selected", selected.Count);

        return selected;
    }

    private void Rebalance()
    {
        // For an equal-weighted portfolio with the Equities that are above their SMA.
        var symbols = _universe.Selected.Where(symbol => Securities[symbol].Price > 0);
        if (symbols.Count() == 0)
        {
            return;
        }
        var weight = 1m / symbols.Count();
        SetHoldings(symbols.Select(symbol => new PortfolioTarget(symbol, weight)).ToList(),
liquidateExistingHoldings: true);
    }
}

// Define a separate class to contain the SMA indicator.
class SelectionData
{
    private SimpleMovingAverage _sma;
    public bool IsAboveSma { get; private set; }

    public SelectionData(QCAlgorithm algorithm, Symbol symbol, int period = 200)
    {
        // Create the SMA indicator for trend detection and filtering.
        _sma = new SimpleMovingAverage(period);
        // Warm up the SMA so you can immediately use it.
        algorithm.WarmUpIndicator(symbol, _sma, Resolution.Daily);
    }

    public void Update(DateTime time, decimal value)
    {
        IsAboveSma = _sma.Update(time, value) && value > _sma.Current.Value;
    }
```

```
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
ETFConstituentUniverseRSIAlphaModelAlgorithm.cs C# UniverseOnlyRegressionAlgorithm.cs C#

Equity

Chained Universes

Introduction

You can combine ("chain") universes together to fetch fundamental and alternative data on a specific subset of assets. Universes filter input data and return `Symbol` objects. The only requirement is that `Symbol` objects the filter returns are a subset of the input data. The source of the `Symbol` objects is unrestricted, so you can feed the output of one universe into another.

Filter Pattern

Universes filter a large set of Symbol objects by a coarse filter to quickly reduce the data processing requirement. This is often a first step before applying a second filter or requesting alternative data. For example, a strategy might only be interested in easily tradable liquid assets so quickly eliminates all stocks with less than $1M USD / day in trading volume.

The order of your filters can improve the speed of your research. By applying filters that narrow the universe the most, or are the lightest weight first, you can significantly reduce the amount of data your algorithm processes. Unless necessary, you can also not return any selections from earlier filters to further improve research speed, keeping only the universe data for later filters.

Universe Data Weights

To speed up your algorithm, request the lightest weight data first before chaining heavier filters or adding alternative data. The following table shows the size each dataset:

Name	Data Size / Weight
US Equities (Fundamental - Dollar Volume only)	Light (100 MB)
US Equities (Fundamental)	Heavy (up to 20 GB)
US Equity Options	Huge (200 TB)
US Index Options	Medium (500 GB)
US Futures	Medium (500 GB)
US Futures Options	Medium (500 GB)
Crypto	Light (1 GB)
Alternative / General	Light (100 MB - 2 GB)
Alternative / Tiingo News	Medium (200 GB)

Chain Fundamental and Alternative Data

The following example chains a fundamental universe and a QuiverCNBCsUniverse alternative universe . It first selects the 100 most liquid US Equities and then filters them down to those mentioned by CNBC commentator/trader Jim Cramer. The output of the alternative universe selection method is the output of the chained universe.

C#

```
public class ChainedUniverseAlgorithm : QCAlgorithm
{
    private List<Symbol> _fundamental = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Filter the top 100 liquid equities of the last trading day, and save the symbols for the next filtering.
        AddUniverse(fundamental =>
        {
            _fundamental = (from c in fundamental
                orderby c.DollarVolume descending
                select c.Symbol).Take(100).ToList();
            return Universe.Unchanged;
        });
        // Filter the equities being commented on by CNBC analyst Cramer, then select the ones that intersect with
the fundamental universe.
        AddUniverse<QuiverCNBCsUniverse>(altCoarse =>
        {
            var followers = from d in altCoarse.OfType<QuiverCNBCsUniverse>()
                where d.Traders.ToLower().Contains("cramer")
                select d.Symbol;
            return _fundamental.Intersect(followers);
        });
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        // Request CNBC data for the selected stocks.
        foreach (var added in changes.AddedSecurities)
        {
            AddData<QuiverCNBCs>(added.Symbol);
        }
    }

    public override void OnData(Slice data)
    {
        foreach (var dataPoint in data.Get<QuiverCNBCs>().SelectMany(x=> x.Value.OfType<QuiverCNBC>()))
        {
            Debug($"{dataPoint.Symbol} traders at {data.Time}: {dataPoint.Traders}");
        }
    }
}
```

Chain Fundamental and US Equity Options

The following example chains a fundamental universe and an Equity Options universe . It first selects 10 stocks with the lowest PE ratio and then selects their front-month call Option contracts. The output of both universes is the output of the chained universe.

```C#
public class ETFUniverseOptions : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        UniverseSettings.Asynchronous = true;
        // Need to set data normalization mode to raw for options to compare the strike price fairly.
        UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;

        // Filter for equities with fundamental data first.
        var universe = AddUniverse(FundamentalFunction);
        // Based on the filtered equities, request an option universe with them as underlying.
        AddUniverseOptions(universe, OptionFilterFunction);
    }

    private IEnumerable<Symbol> FundamentalFunction(IEnumerable<Fundamental> fundamental)
    {
        // Filter for equities with the lowest PE Ratio first.
        return fundamental
            .Where(f => !double.IsNaN(f.ValuationRatios.PERatio))
            .OrderBy(f => f.ValuationRatios.PERatio)
            .Take(10)
            .Select(x => x.Symbol);
    }

    private OptionFilterUniverse OptionFilterFunction(OptionFilterUniverse optionFilterUniverse)
    {
        // Select the front month ATM calls.
        return optionFilterUniverse.Strikes(0, 2).FrontMonth().CallsOnly();
    }

    public override void OnData(Slice data)
    {
        // Iterate through each Option chain.
        foreach (var (canonicalSymbol, chain) in data.OptionChains)
        {
            var equitySymbol = canonicalSymbol.Underlying;
            foreach (var contract in chain)
            {
                var contractSymbol = contract.Symbol;
                var price = contract.LastPrice;
            }
        }
    }
}
```

Chain ETF and Fundamental

The following example chains a fundamental universe and an ETF constituents universe . It first selects all the constituents of the QQQ ETF and then filters them down to select the 10 assets with the lowest PE ratio. The output of the fundamental universe selection method is the output of the chained universe.

```C#
public class ChainedUniverseAlgorithm : QCAlgorithm
{
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        UniverseSettings.Asynchronous = true;
        // Select QQQ constituents first, then by fundamental data.
        _universe = AddUniverse(Universe.ETF("QQQ", Market.USA, UniverseSettings, ETFConstituentsFilter),
FundamentalSelection);
    }

    private IEnumerable<Symbol> ETFConstituentsFilter(IEnumerable<ETFConstituentUniverse> constituents)
    {
        // Select all QQQ constituents.
        return constituents.Select(c => c.Symbol);
    }

    private IEnumerable<Symbol> FundamentalSelection(IEnumerable<Fundamental> fundamental)
    {
        // Filter for the lowest PE Ratio stocks among QQQ constituents for undervalued stocks.
        return fundamental
            .Where(f => !double.IsNaN(f.ValuationRatios.PERatio))
            .OrderBy(f => f.ValuationRatios.PERatio)
            .Take(10)
            .Select(x => x.Symbol);
    }

    public override void OnData(Slice data)
    {
        foreach (var symbol in _universe.Selected)
        {
            var peRatio = Securities[symbol].Fundamentals.ValuationRatios.PERatio;
        }
    }
}
```

Chain ETF and Alternative Data

The following example chains an ETF universe and a QuiverCNBCsUniverse alternative universe . It first selects all constituents of SPY and then filters them down to those mentioned by CNBC commentator/trader Jim Cramer. The output of the alternative universe selection method is the output of the chained universe.

```C#
public class ChainedUniverseAlgorithm : QCAlgorithm
{
    private List<Symbol> _etf = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        UniverseSettings.Asynchronous = true;

        // Save all SPY constituents for the next filtering.
        AddUniverse(Universe.ETF("SPY", Market.USA, UniverseSettings, constituents =>
        {
            _etf = constituents.Select(c => c.Symbol).ToList();
            return Universe.Unchanged;
        }));
        // Filter the equities being commented on by CNBC analyst Cramer, then select the ones in SPY constituents.
        AddUniverse<QuiverCNBCsUniverse>(altCoarse =>
        {
            var followers = from d in altCoarse.OfType<QuiverCNBCsUniverse>()
                where d.Traders.ToLower().Contains("cramer")
                select d.Symbol;
            return _etf.Intersect(followers);
        });
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        // Request CNBC data for the selected stocks.
        foreach (var added in changes.AddedSecurities)
        {
            AddData<QuiverCNBCs>(added.Symbol);
        }
    }

    public override void OnData(Slice data)
    {
        foreach (var dataPoint in data.Get<QuiverCNBCs>().SelectMany(x=> x.Value.OfType<QuiverCNBC>()))
        {
            Debug($"{dataPoint.Symbol} traders at {data.Time}: {dataPoint.Traders}");
        }
    }
}
```

Chain ETF and US Equity Options

The following example chains an ETF constituents universe and an Equity Options universe . It first selects the 10 largest-weighted constituents of QQQ and then selects their front-month call Option contracts. The output of both universes is the output of the chained universe.

C#

```
public class ETFUniverseOptions : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        UniverseSettings.Asynchronous = true;
        // Need to set the data normalization mode to raw for fair strike price comparison.
        UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;

        // First-pass filtering from QQQ constituents.
        var etfUniverse = Universe.ETF("QQQ", Market.USA, UniverseSettings, ETFConstituentsFilter);
        AddUniverse(etfUniverse);
        // Filter for equity options from the selected QQQ constituents next.
        AddUniverseOptions(etfUniverse, OptionFilterFunction);
    }

    private IEnumerable<Symbol> ETFConstituentsFilter(IEnumerable<ETFConstituentUniverse> constituents)
    {
        // Select the top 10 weighted QQQ constituents for most liquid options.
        return constituents.OrderByDescending(c => c.Weight).Take(10).Select(c => c.Symbol);
    }

    private OptionFilterUniverse OptionFilterFunction(OptionFilterUniverse optionFilterUniverse)
    {
        // Select the front month ATM standard calls.
        return optionFilterUniverse.Strikes(0, 2).FrontMonth().CallsOnly();
    }

    public override void OnData(Slice data)
    {
        // Iterate through each Option chain.
        foreach (var (canonicalSymbol, chain) in data.OptionChains)
        {
            var equitySymbol = canonicalSymbol.Underlying;
            foreach (var contract in chain)
            {
                var contractSymbol = contract.Symbol;
                var price = contract.LastPrice;
            }
        }
    }
}
```

Equity

Legacy Fundamental Universes

Introduction

Warning: This API for Universe Selection was deprecated on November 2023. Please refer to the new Fundamental Universe API .

There are several ways to create an Equities universe. You can select a universe based on `CoarseFundamental` data or the constituents of an ETF, and then you can further filter your universe down with corporate fundamentals. The following sections explain each of these techniques in detail.

Coarse Universe Selection

A coarse universe enables you pick a set of stocks based on their trading volume, price, or whether they have fundamental data. To add a coarse universe, in the `Initialize` method, pass a filter function to the `AddUniverse` method. The coarse filter function receives a list of `CoarseFundamental` objects and must return a list of `Symbol` objects. The `Symbol` objects you return from the function are the constituents of the universe and LEAN automatically creates subscriptions for them. Don't call `AddEquity` in the filter function.

```
C#
public class MyCoarseUniverseAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        UniverseSettings.Asynchronous = true;
        AddUniverse(CoarseFilterFunction);
    }

    private IEnumerable<Symbol> CoarseFilterFunction(IEnumerable<CoarseFundamental> coarse)
    {
        return (from c in coarse
            orderby c.DollarVolume descending
            select c.Symbol).Take(100);
    }
}
```

`CoarseFundamental` objects have the following attributes:

The total number of stocks in the US Equity Security Master dataset is 30,000 but your coarse filter function won't receive all of these at one time because the US Equity Security Master dataset is free of survivorship bias and some of the securities have delisted over time. The number of securities that are passed into your coarse filter function depends on the date of your algorithm. Currently, there are about 10,000 securities that LEAN passes into your coarse filter function.

Fundamentals Selection

A fundamental universe lets you select stocks based on corporate fundamental data. This data is powered by Morningstar® and includes approximately 8,100 tickers with 900 properties each. This universe doesn't include ETFs, ADRs, or OTC equities. Due to the sheer volume of information, fundamental selection is performed on the output of another universe filter. Think of this process as a 2-stage filter. An initial filter function selects a set of stocks and then a fine fundamental filter function selects a subset of those stocks.

QuantConnect Coarse and Fine Universe Selection

To add a fundamental universe, in the `Initialize` method, pass two filter functions to the `AddUniverse` method. The first filter function can be a coarse universe filter , dollar volume filter , or an ETF constituents filter . The second filter function receives a list of `FineFundamental` objects and must return a list of `Symbol` objects. The list of `FineFundamental` objects contains a subset of the `Symbol` objects that the first filter function returned. The `Symbol` objects you return from the second function are the constituents of the fundamental universe and LEAN automatically creates subscriptions for them. Don't call `AddEquity` in the filter function.

> **Tip:** Only 8,100 assets have fundamental data. If your first filter function receives `CoarseFundamental` data, you should only select assets that have a true value for their `HasFundamentalData` property.

C#

```
public class MyUniverseAlgorithm : QCAlgorithm {
    public override void Initialize()
    {
        UniverseSettings.Asynchronous = true;
        AddUniverse(CoarseFilterFunction, FineFundamentalFilterFunction);
    }
    // filter based on CoarseFundamental
    IEnumerable<Symbol> CoarseFilterFunction(IEnumerable<CoarseFundamental> coarse)
    {
         // In addition to further coarse universe selection, ensure the security has fundamental data
         return (from c in coarse
             where c.HasFundamentalData
             select c.Symbol);
    }
    // filter based on FineFundamental
    public IEnumerable<Symbol> FineFundamentalFilterFunction(IEnumerable<FineFundamental> fine)
    {
        // Return a list of Symbols
        return Enumerable.Empty<Symbol>();
    }
}
```

`FineFundamental` objects have the following attributes:

Example

The simplest example of accessing the fundamental object would be harnessing the iconic PE ratio for a stock. This is a ratio of the price it commands to the earnings of a stock. The lower the PE ratio for a stock, the more affordable it appears.

```
// Take the top 50 by dollar volume using coarse
// Then the top 10 by PERatio using fine
UniverseSettings.Asynchronous = true;
AddUniverse(
    coarse => {
        return (from c in coarse
            where c.Price > 10 && c.HasFundamentalData
            orderby c.DollarVolume descending
            select c.Symbol).Take(50);
    },
    fine => {
        return (from f in fine
            orderby f.ValuationRatios.PERatio ascending
            select f.Symbol).Take(10);
    });
```

Asset Categories

In addition to valuation ratios, the US Fundamental Data from Morningstar has many other data point attributes, including over 200 different categorization fields for each US stock. Morningstar groups these fields into sectors, industry groups, and industries.

Sectors are large super categories of data. To get the sector of a stock, use the `MorningstarSectorCode` property.

```
var tech = fine.Where(x => x.AssetClassification.MorningstarSectorCode == MorningstarSectorCode.Technology);
```

Industry groups are clusters of related industries that tie together. To get the industry group of a stock, use the `MorningstarIndustryGroupCode` property.

```
var ag = fine.Where(x => x.AssetClassification.MorningstarIndustryGroupCode ==
MorningstarIndustryGroupCode.Agriculture);
```

Industries are the finest level of classification available. They are the individual industries according to the Morningstar classification system. To get the industry of a stock, use the `MorningstarIndustryCode` .

```
var coal = fine.Where(x => x.AssetClassification.MorningstarIndustryCode == MorningstarSectorCode.Coal);
```

Practical Limitations

Like coarse universes, fine universes allow you to select an unlimited universe of assets to analyze. Each asset in the universe consumes approximately 5MB of RAM, so you may quickly run out of memory if your universe filter selects many assets. If you backtest your algorithms in the Algorithm Lab, familiarize yourself with the RAM capacity of your backtesting and live trading nodes . To keep your algorithm fast and efficient, only subscribe to the assets you need.

Live Trading Considerations

The live data for fundamental universe selection arrives at 6/7 AM Eastern Time (ET), so fundamental universe selection runs for live algorithms between 7 and 8 AM ET. This timing allows you to place trades before the market opens. Don't schedule anything for midnight because the universe selection data isn't ready yet.

Examples

The legacy fundamental universe is deprecated. For examples of the new fundamental universe API, see Examples .

Equity

Alternative Data Universes

Introduction

An alternative data universe lets you select a basket of Equities based on an alternative dataset that's linked to them. If you use an alternative data universe, you limit your universe to only the securities in the dataset, which avoids unnecessary subscriptions.

Supported Datasets

The following alternative datasets support universe selection:

- Brain Language Metrics on Company Filings
- Brain ML Stock Ranking
- Brain Sentiment Indicator
- Crypto Market Cap
- Upcoming Dividends
- Upcoming Earnings
- Upcoming IPOs
- Upcoming Splits
- CNBC Trading
- Corporate Lobbying
- Insider Trading
- US Congress Trading
- US Government Contracts
- WallStreetBets
- Corporate Buybacks

Examples

The following examples demonstrate some common alternative data universes.

Example 1: Brain Sentiment Universe

The following algorithm uses the Brain Sentiment Indicator dataset to create a universe of US Equities that have some article mentions and the most positive sentiment. It then forms a sentiment-weighted portfolio at the start of each month.

```C#
public class BrainSentimentUniverseAlgorithm : QCAlgorithm
{
    private Dictionary<Symbol, decimal> _sentimentBySymbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add an Equity universe that updates at the start of each month.
        var spy = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
        UniverseSettings.Schedule.On(DateRules.MonthStart(spy));
        AddUniverse<BrainSentimentIndicatorUniverse>(altData =>
        {
            _sentimentBySymbol = altData
                .Select(x => x as BrainSentimentIndicatorUniverse)
                // Select the assets with recent mentions and a sentiment score.
                .Where(x => x.TotalArticleMentions30Days > 0 && x.Sentiment30Days.HasValue)
                // Select the 10 assets with the greatest sentiment score.
                .OrderByDescending(x => x.Sentiment30Days.Value)
                .Take(10)
                .ToDictionary(x => x.Symbol, x => x.Sentiment30Days.Value);
            // Return the Symbol objects of the selected assets.
            return _sentimentBySymbol.Keys;
        });
        // Create a Scheduled Event to rebalance the portfolio at the start of each month.
        Schedule.On(DateRules.MonthStart(spy), TimeRules.At(9, 45), Rebalance);
    }

    public void Rebalance()
    {
        // To avoid trading errors, filter out the assets that don't have a price yet.
        _sentimentBySymbol = _sentimentBySymbol
            .Where(kvp => Securities[kvp.Key].Price > 0)
            .ToDictionary(kvp => kvp.Key, kvp => kvp.Value);
        // Form a sentiment-weighted portfolio.
        var sentimentScoreSum = _sentimentBySymbol.Values.Sum();
        SetHoldings(
            _sentimentBySymbol.Select(kvp => new PortfolioTarget(kvp.Key, kvp.Value / sentimentScoreSum)).ToList(),
            liquidateExistingHoldings: true
        );
    }
}
```

Example 2: Insiders Trading Universe

Insiders have more information to evaluate the overall prospect of the company, so following their trades can be useful. The following algorithm uses the Insider Trading to create a universe of US Equities that insiders have recently purchased. It then invests equally into the companies with positive insider trades, which may provide extra confidence in their expected return, and hold for 3 months.

C#

```
public class InsiderTradingUniverseAlgorithm : QCAlgorithm
{
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Seed the price of each asset with its last known price to
        // avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Keep each security in the universe for a minimum of 30 days to
        // digest the insiders purchase sentiment.
        UniverseSettings.MinimumTimeInUniverse = TimeSpan.FromDays(30);
        // Add the QuiverInsiderTrading dataset for insider trade
        // detection and filtering.
        _universe = AddUniverse<QuiverInsiderTradingUniverse>(SelectAssets);
        // Add a Scheduled Event to rebalance the portfolio each day.
        var spy = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
        Schedule.On(DateRules.EveryDay(spy), TimeRules.AfterMarketOpen(spy, 1), Rebalance);
    }

    private IEnumerable<Symbol> SelectAssets(IEnumerable<BaseData> altData)
    {
        // Calculate the net dollar volume of insider trades for each
        // asset.
        var dollarVolumeBySymbol = new Dictionary<Symbol, decimal?>();
        foreach (QuiverInsiderTradingUniverse data in altData)
        {
            if (data.PricePerShare == 0m || data.PricePerShare == null)
            {
                continue;
            }
            if (!dollarVolumeBySymbol.ContainsKey(data.Symbol))
            {
                dollarVolumeBySymbol[data.Symbol] = 0;
            }
            dollarVolumeBySymbol[data.Symbol] += data.Shares * data.PricePerShare;
        }
        // Select the 10 assets with the largest dollar volume of insider
        // trades.
        return dollarVolumeBySymbol
            .OrderByDescending(kvp => kvp.Value)
            .Take(10)
            .Select(kvp => kvp.Key);
    }

    public void Rebalance()
    {
        // Equally invest in insider buying companies to evenly
        // dissipate the capital risk.
        var symbols = _universe.Selected.Where(symbol => Securities[symbol].Price != 0);
        SetHoldings(symbols.Select(s => new PortfolioTarget(s, 1m / symbols.Count())).ToList(), true);
    }
}
```

Example 3: Share Buyback Universe

The following algorithm uses the Corporate Buybacks dataset to create a universe of US Equities that have announced an upcoming share buyback program:

C#

```
public class SmartInsiderIntentionUniverseAlgorithm : QCAlgorithm
{
    // A dictionary to hold the buyback size for position sizing.
    public Dictionary<Symbol, decimal> _buybackSize = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Seed the price of each asset with its last known price to
        // avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Add a universe that selects assets at the start of each week.
        // The universe should select Equities that have announced a
        // material buyback plan, since they have confidence in their
        // future prospect and the reduction in supply can drive their
        // price up.
        UniverseSettings.Schedule.On(DateRules.WeekStart());
        AddUniverse<SmartInsiderIntentionUniverse>(altData => {
            _buybackSize.Clear();
            return altData.OfType<SmartInsiderIntentionUniverse>()
                // Select assets that have a material buyback size to have
                // strong confidence.
                .Where(d => d.AmountValue.HasValue && d.AmountValue.Value > 10000000m)
                .Select(d => {
                    // Record the buyback size in the dictionary.
                    _buybackSize[d.Symbol] = d.AmountValue.Value;
                    return d.Symbol;
                });
        });
        // Add a Scheduled Event to rebalance the portfolio at the start
        // of each week. The weekly schedule allows time to capitalize
        // the market swings from the positive sentiment.
        var spy = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
        Schedule.On(DateRules.WeekStart(spy), TimeRules.AfterMarketOpen(spy, 1), Rebalance);
    }

    private void Rebalance()
    {
        // Ignore illiquid assets that have no price yet.
        _buybackSize = _buybackSize.Where(kvp => Securities[kvp.Key].Price != 0).ToDictionary();
        // Get the sum of all buyback size so we can scale the portfolio
        // weights based on the buyback size.
        var buybackSum = _buybackSize.Sum(x => x.Value);
        // Place the trades to rebalance the portfolio.
        SetHoldings(_buybackSize.Select(x => new PortfolioTarget(x.Key, x.Value / buybackSum)).ToList());
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        // Liquidate assets when they leave the universe.
        foreach (var removed in changes.RemovedSecurities)
        {
            _buybackSize.Remove(removed.Symbol);
            Liquidate(removed.Symbol);
        }
    }
}
```

Universes

Equity Options

Introduction

An Equity Options universe lets you select a basket of contracts for a single Option. LEAN models Option subscriptions as a universe of Option contracts.

Create Universes

To add a universe of Equity Option contracts, in the `Initialize` method, call the `AddOption` method. This method returns an `Option` object, which contains the canonical `Symbol` . You can't trade with the canonical Option `Symbol` , but save a reference to it so you can easily access the Option contracts in the OptionChain that LEAN passes to the `OnData` method.

C#

```
public class BasicOptionAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        UniverseSettings.Asynchronous = true;
        var option = AddOption("SPY");
        option.SetFilter(Filter);
        _symbol = option.Symbol;
    }

    private OptionFilterUniverse Filter(OptionFilterUniverse universe)
    {
        return universe.IncludeWeeklys().Expiration(0, 7).Delta(0.35m, 0.75m);
    }

    public override void OnData(Slice data)
    {
        if (data.OptionChains.TryGetValue(_symbol, out var chain))
        {
            var contract = chain.OrderBy(x => x.Greeks.Delta).FirstOrDefault();
        }
    }
}
```

The following table describes the `AddOption` method arguments:

Argument	Data Type	Description	Default Value
`ticker`	`string`	The underlying Equity ticker. To view the supported underlying Equity tickers, see Supported Assets .	
`resolution`	`Resolution?`	The resolution of the market data. To view the supported resolutions, see Resolutions . The Equity resolution must be less than or equal to the Equity Option resolution. For example, if you set the Equity resolution to minute, then you must set the Equity Option resolution to minute, hour, or daily.	`null`
`market`	`string`	The underlying Equity market.	`null`
`fillForward`	`bool`	If true, the current slice contains the last available data even if there is no data at the current time.	`true`
`leverage`	`decimal`	The leverage for this Equity.	`Security.NullLeverage`
`extendedMarketHours`	`bool`	A flag that signals if LEAN should send data during pre- and post-market trading hours.	`false`

If you add an Equity Option universe but don't have a subscription to the underlying Equity, LEAN automatically subscribes to the underlying Equity with the following settings:

Setting	Value
Fill forward	Same as the Option universe
Leverage	0
Extended Market Hours	Same as the Option universe
Data Normalization	`DataNormalizationMode.Raw`

If you already have a subscription to the underlying Equity but it's not `Raw` data normalization, LEAN automatically changes it to `Raw` .

Filter by Investment Strategy

Options trading strategies consist of simultaneously buying or selling one or more Option contracts of the same underlying Equity, but with different features.

```C#
// Example 1: Select a Straddle
option.SetFilter(optionFilterUniverse => optionFilterUniverse.Straddle(30, 5, 10));

// Example 2: Select the contracts (including weeklys) that expire in the next 30 days that form an Iron Condor
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().Strikes(-20, 20).Expiration(0,
30).IronCondor(30, 5, 10));

// Example 3: Select the 0DTE contracts that form an Strangle
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().Expiration(0, 0).Strangle(30, 5, -
10));
```

The following table describes the filter methods of the `OptionFilterUniverse` class that select contracts for Option strategies:

`NakedCall(int minDaysTillExpiry, decimal strikeFromAtm)` Selects a call contract to form Naked Call , Covered Call , or Protective Call Option strategies.
`NakedPut(int minDaysTillExpiry, decimal strikeFromAtm)` Selects a put contract to form Naked Put , Covered Put , or Protective Put Option strategies.
`CallSpread(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects two call contracts to form Bull Call Spread or Bear Call Spread Option strategies.
`PutSpread(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects two put contracts to form Bull Put Spread or Bear Put Spread Option strategies.
`CallCalendarSpread(decimal strikeFromAtm, int minNearDaysTillExpiry, int minFarDaysTillExpiry)` Selects two call contracts to form Long Call Calendar Spread or Short Call Calendar Spread Option strategies.
`PutCalendarSpread(decimal strikeFromAtm, int minNearDaysTillExpiry, int minFarDaysTillExpiry)` Selects two put contracts to form Long Put Calendar Spread or Short Put Calendar Spread Option strategies.
`Strangle(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects two contracts to form Long Strangle or Short Strangle Option strategies.
`Straddle(int minDaysTillExpiry)` Selects two contracts to form Long Straddle or Short Straddle Option strategies.
`ProtectiveCollar(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects two contracts to form Protective Collar Option strategies.

`Conversion(int minDaysTillExpiry, decimal strikeFromAtm)` Selects two contracts to form Conversion or Reverse Conversion Option strategies.
`CallButterfly(int minDaysTillExpiry, decimal strikeSpread)` Selects three contracts to form Long Call Butterfly or Short Call Butterfly Option strategies.
`PutButterfly(int minDaysTillExpiry, decimal strikeSpread)` Selects three contracts to form Long Put Butterfly or Short Put Butterfly Option strategies.
`IronButterfly(int minDaysTillExpiry, decimal strikeSpread)` Selects four contracts to form Long Iron Butterfly or Short Iron Butterfly Option strategies.
`IronCondor(int minDaysTillExpiry, decimal nearStrikeSpread, decimal farStrikeSpread)` Selects four contracts to form Long Iron Condor or Short Iron Condor Option strategies.
`BoxSpread(int minDaysTillExpiry, decimal strikeSpread)` Selects four contracts to form Box Spread or Short Box Spread Option strategies.
`JellyRoll(decimal strikeFromAtm, int minNearDaysTillExpiry, int minFarDaysTillExpiry)` Selects four contracts to form Jelly Roll or Short Jelly Roll Option strategies.
`CallLadder(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal middleStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects four contracts to form Bear Call Ladder or Bull Call Ladder Option strategies.
`PutLadder(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal middleStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects four contracts to form Bear Put Ladder or Bull Put Ladder Option strategies.

The preceding methods return an `OptionFilterUniverse` , so you can chain the methods together.

Filter by Implied Volatility and Greeks

Option price models compute the theoretical price of Option contracts, their implied volatility, and their Greek values. Theoretical prices can help you detect undervalued and overvalued contracts, implied volatility can provide you insight into the upcoming volatility of the underlying security, and Greek values can help you hedge your portfolio.

```
// Example 1: Select the contracts with delta between 0.25 and 0.75
option.SetFilter(optionFilterUniverse => optionFilterUniverse.Delta(0.25m, 0.75m));

// Example 2: Select the contracts (including weeklys) that expire in the next 90 days with implied volatility below
20%
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().IV(0, 20).Expiration(0, 90));

// Example 3: Select the contracts (including weeklys) that expire in the next 30 days with implied volatility below
20% that forms an Iron Condor
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().IV(0, 20).Expiration(0,
30).IronCondor(30, 5, 10));
```

The following table describes the filter methods of the `OptionFilterUniverse` class that select contract for a range of implied volatility, Greek values, and open interest:

`IV(decimal min, decimal max)`
`ImpliedVolatility(decimal min, decimal max)`
 	
Selects a contract with implied volatility within the range you set.

`D(decimal min, decimal max)`
`Delta(decimal min, decimal max)`
 	
Selects a contract with delta within the range you set.

`G(decimal min, decimal max)`
`Gamma(decimal min, decimal max)`
 	
Selects a contract with gamma within the range you set.

`R(decimal min, decimal max)`
`Rho(decimal min, decimal max)`
 	
Selects a contract with rho within the range you set.

`V(decimal min, decimal max)`
`Vega(decimal min, decimal max)`
 	
Selects a contract with vega within the range you set.

`T(decimal min, decimal max)`
`Theta(decimal min, decimal max)`
 	
Selects a contract with theta within the range you set.

`OI(long min, long max)`
`OpenInterest(decimal min, decimal max)`
 	
Selects a contract with open interest within the range you set.

The preceding methods return an `OptionFilterUniverse` , so you can chain the methods together.

To perform thorough filtering on the `OptionFilterUniverse` , define an isolated filter method.

```C#
// Select the put contracts with the lowest strike price.
option.SetFilter(Selector);

private OptionFilterUniverse Selector(OptionFilterUniverse optionFilterUniverse)
{
    return universe
        .Delta(0.5m, 1.5m)
        .Gamma(0.0001m, 0.0006m)
        .Vega(0.01m, 1.5m)
        .Theta(-2.0m, -0.5m)
        .Rho(0.5m, 3.0m)
        .ImpliedVolatility(1.0m, 3.0m)
        .OpenInterest(100, 500);
}
```

Filter by Other Contract Properties

To set a contract filter, in the `Initialize` method, call the `SetFilter` method of the `Option` object.

```C#
// Select contracts that have a strike price within 1 strike level above and below the underlying price
option.SetFilter(minStrike: -1, maxStrike: 1);

// Select contracts that expire within 30 days
option.SetFilter(minExpiry: TimeSpan.FromDays(0), maxExpiry: TimeSpan.FromDays(30));

// Select contracts that have a strike price within 1 strike level and expire within 30 days
option.SetFilter(minStrike: -1, maxStrike: 1, minExpiry: TimeSpan.FromDays(0), maxExpiry: TimeSpan.FromDays(30));

// Select call contracts
option.SetFilter(optionFilterUniverse => optionFilterUniverse.CallsOnly());
```

The following table describes the available filter techniques:

`SetFilter(int minStrike, int maxStrike)` Selects the contracts that have a strike price within a minimum and maximum strike level relative to the underlying price. For example, say the underlying price is 302*andtherearestrikesatevery*5. If you set `minStrike` to -1 and `maxStrike` to 1, LEAN selects the contracts that have a strike of 300*or*305. This filter runs asynchronously by default.
`SetFilter(TimeSpan minExpiry, TimeSpan maxExpiry)` Selects the contracts that expire within the range you set. This filter runs asynchronously by default.
`SetFilter(int minStrike, int maxStrike, TimeSpan minExpiry, TimeSpan maxExpiry)` Selects the contracts that expire and have a strike within the range you set. This filter runs asynchronously by default.
`SetFilter(Func<OptionFilterUniverse, OptionFilterUniverse> universeFunc)` Selects the contracts that a function selects.

The following table describes the filter methods of the `OptionFilterUniverse` class:

`Strikes(int minStrike, int maxStrike)` Selects contracts that are within `minStrike` strikes below the underlying price and `maxStrike` strikes above the underlying price.
`CallsOnly()` Selects call contracts.
`PutsOnly()` Selects put contracts.
`StandardsOnly()` Selects standard contracts.
`IncludeWeeklys()` Selects non-standard weeklys contracts.
`WeeklysOnly()` Selects weekly contracts.
`FrontMonth()` Selects the front month contract.
`BackMonths()` Selects the non-front month contracts.
`BackMonth()` Selects the back month contracts.
`Expiration(int minExpiryDays, int maxExpiryDays)` Selects contracts that expire within a range of dates relative to the current day.
`Contracts(IEnumerable<Symbol> contracts)` Selects a list of contracts.
`Contracts(Func<IEnumerable<Symbol>, IEnumerable< Symbol>> contractSelector)` Selects contracts that a selector function selects.

The preceding methods return an `OptionFilterUniverse` , so you can chain the methods together.

```C#
// Example 1: Select the front month call contracts
option.SetFilter(optionFilterUniverse => optionFilterUniverse.CallsOnly().FrontMonth());

// Example 2: Select the contracts (including weeklys) that expire in the next 90 days
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().Strikes(-20, 20).Expiration(0, 90));

// Example 3: Select the contracts (including weeklys) that expire in the next 30 days that form an Iron Condor
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().Strikes(-20, 20).Expiration(0,
30).IronCondor(30, 5, 10));
```

Some of the preceding filter methods only set an internal enumeration in the `OptionFilterUniverse` that it uses later on in the filter process. This subset of filter methods don't immediately reduce the number of contract `Symbol` objects in the `OptionFilterUniverse` .

Default Filter

By default, LEAN subscribes to the Option contracts that have the following characteristics:

- Standard type (exclude weeklys)
- Within 1 strike price of the underlying asset price
- Expire within 35 days

To adjust the universe of contracts, set a filter. The filter usually runs at the first bar of every day. When the filter selects a contract that isn't currently in your universe, LEAN adds the new contract data to the next `Slice` that it passes to the `OnData` method.

Navigate Intraday Option Chains

`OptionChain` objects represent an entire chain of Option contracts for a single underlying security.

To get the `OptionChain` , index the `OptionChains` property of the `Slice` with the canonical `Symbol` . After you get the `OptionChain` , you can sort and filter the Option contracts in the chain.

```C#
public override void OnData(Slice slice)
{
    // Try to get the OptionChain using the canonical symbol
    if (slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        // Example: Find 5 put contracts that are closest to at-the-money (ATM) and have the farthest expiration
        var contracts = chain
            .Where(x => x.Right == OptionRight.Put)
            .OrderByDescending(x => x.Expiry)
            .ThenBy(x => Math.Abs(chain.Underlying.Price - x.Strike))
            .Take(5);

        // Select the contract with the delta closest to -0.5
        var contract = contracts.OrderBy(x => Math.Abs(-0.5m - x.Greeks.Delta)).FirstOrDefault();
    }
}
```

You can also loop through the `OptionChains` property to get each `OptionChain` .

```C#
public override void OnData(Slice slice)
{
    // Iterate all received Canonical Symbol-OptionChain key-value pairs
    foreach (var kvp in slice.OptionChains)
    {
        var canoncialSymbol = kvp.Key;
        var chain = kvp.Value;
        var contracts = chain.Contracts;
    }
}
```

`OptionChain` objects have the following properties:

Navigate Daily Option Chains

To get the daily, pre-calculated Greeks and implied volaility of all the currently tradable contracts, call the `OptionChain` method. This method returns a collection of `OptionUniverse` objects, where each object represents a contract. The data this method returns contains information on all the currently tradable contracts, not just the contracts that pass your filter.

```C#
// In the initialize method, create a Scheduled Event to get the Option
// chain and rebalance the portfolio.
Schedule.On(DateRules.WeekStart(_symbol), TimeRules.AfterMarketOpen(_symbol, 1), Rebalance);

public void Rebalance()
{
    foreach (var optionUniverse in OptionChain(_symbol))
    {
        var close = optionUniverse.Close;
        var oi = optionUniverse.OpenInterest;
        var delta = optionUniverse.Greeks.Delta;
    }
}
```

The `OptionChain` method was previously known as the `OptionChainProvider.GetOptionContractList` method.

`OptionUniverse` objects have the following properties:

Greeks and Implied Volatility

There are several ways to get the implied volatility and Greeks of contracts.

The Greeks and IV values in the filter function are the daily, pre-calculated values based on the end of the previous trading day.

```C#
public override void Initialize()
{
    var option = AddOption("SPY");
    option.SetFilter(universe => universe.IncludeWeeklys().Delta(0.3m, 0.7m).Expiration(0, 7));
}
```

To calculate the values, we use our implementation of the forward tree pricing model, which accounts for the interest rate and dividend payments.

You can't customize the Greeks and IV values that the filter function receives. However, you can create indicators to customize how the Greeks and IV are calculated for the contracts already in your universe.

To override the default pricing model of the Option, set a pricing model . If you set a price model, it sets the price model of the individual contracts in the universe.

```
// Set PriceModel property to use the Crank-Nicolson finite-difference model to price the Options.
option.PriceModel = OptionPriceModels.CrankNicolsonFD();
```

To override the initial guess of implied volatility , set and warm up the underlying volatility model .

Historical Data

To get historical chains for an Equity Option, call the `History<OptionUniverse>` method with the canonical Option `Symbol` . This method returns the entire Option chain for each trading day, not the subset of contracts that pass your universe filter.

```
public class OptionHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        var option = AddOption("SPY");
        var history = History<OptionUniverse>(option.Symbol, 5);
        foreach (var chain in history)
        {
            var endTime = chain.EndTime;
            var filteredContracts = chain.Data
                .Select(contract => contract as OptionUniverse)
                .Where(contract => contract.Greeks.Delta > 0.3m);
            foreach (var contract in filteredContracts)
            {
                var price = contract.Price;
                var iv = contract.ImpliedVolatility;
            }
        }
    }
}
```

The Greeks and IV values that you get from a history request of the Option universe are the daily, pre-calculated values based on the end of the previous trading day. To get the intraday values or to customize the Greeks and IV calculations, create some Option indicators .

The `History<OptionUniverse>` method represents each contract with an `OptionUniverse` object, which have the following properties:

Selection Frequency

By default, Equity Option universes run at the first time step of each day to select their contracts.

Examples

The following examples demonstrate some common Equity Option universes.

Example 1: 0DTE Contracts

0DTE Options are Option contracts that expire on the same day you trade them. The following algorithm selects 0DTE Option contracts for the SPY that fall within 3 strikes of the underlying price.

```
C#
public class ZeroDTEOptionUniverseAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        var option = AddOption("SPY");
        option.SetFilter(u => u.IncludeWeeklys().Expiration(0, 0).Strikes(-3, 3));
    }
}
```

For trading full examples of Equity Option Universe, please refer to the Securities > Equity Options > Requesting Data > Universes page.

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
OptionEquityBearCallLadderRegressionAlgorithm.cs C# OptionEquityBearCallSpreadRegressionAlgorithm.cs C#
OptionEquityBearPutLadderRegressionAlgorithm.cs C# OptionEquityBearPutSpreadRegressionAlgorithm.cs C#
OptionEquityBoxSpreadRegressionAlgorithm.cs C# OptionEquityBullCallLadderRegressionAlgorithm.cs C#
OptionEquityBullCallSpreadRegressionAlgorithm.cs C# OptionEquityBullPutLadderRegressionAlgorithm.cs C#
OptionEquityBullPutSpreadRegressionAlgorithm.cs C# OptionEquityCallButterflyRegressionAlgorithm.cs C#
OptionEquityCallCalendarSpreadRegressionAlgorithm.cs C# OptionEquityConversionRegressionAlgorithm.cs C#
OptionEquityCoveredCallRegressionAlgorithm.cs C# OptionEquityCoveredPutRegressionAlgorithm.cs C#
OptionEquityIronButterflyRegressionAlgorithm.cs C# OptionEquityIronCondorRegressionAlgorithm.cs C#
OptionEquityJellyRollRegressionAlgorithm.cs C# OptionEquityProtectiveCollarRegressionAlgorithm.cs C#
OptionEquityPutButterflyRegressionAlgorithm.cs C# OptionEquityPutCalendarSpreadRegressionAlgorithm.cs C#
OptionEquityReverseConversionRegressionAlgorithm.cs C# OptionEquityShortBoxSpreadRegressionAlgorithm.cs C#
OptionEquityShortButterflyCallRegressionAlgorithm.cs C# OptionEquityShortButterflyPutRegressionAlgorithm.cs C#
OptionEquityShortIronButterflyRegressionAlgorithm.cs C# OptionEquityShortIronCondorRegressionAlgorithm.cs C#
OptionEquityShortJellyRollRegressionAlgorithm.cs C# OptionEquityStraddleRegressionAlgorithm.cs C#
OptionEquityStrangleRegressionAlgorithm.cs C#

Universes

Equity CFD

Introduction

An Equity CFD universe lets you select a set of Equity CFD assets. To select the underlying Equities, you can use a fundamental universe to select assets based on corporate fundamental data, an ETF constituents universe to select the CFD versions of stocks in an exchange-traded fund, or an alternative data universe to select assets based on alternative data. This feature is powerful for European clients seeking to trade US ETF products.

Only the Interactive Brokers CFD integration supports trading Stock-CFD products. QuantConnect doesn't have historical data for Interactive Brokers CFD products; however, you can use the `LiveMode` flag to swap to the CFD equivalents for live trading. In live trading, you must include Interactive Brokers as a data provider to trade Equity-CFD products.

Create Universes Using Fundamentals

To add a fundamental universe, in the `Initialize` method, pass a filter function to the `AddUniverse` method. The filter function receives a list of `Fundamental` objects and must return a list of `Symbol` objects. The `Symbol` objects you return from the function are the constituents of the fundamental universe and LEAN automatically creates subscriptions for them. In live mode, call the `Symbol.Create` method to swap for a CFD version of the same `Symbol` . Don't call `AddCfd` in the filter function.

C#

```
// Create a universe for CFD securities based on fundamental data.
public class MyUniverseAlgorithm : QCAlgorithm {
    private Universe _universe;
    public override void Initialize()
    {
        UniverseSettings.Asynchronous = true;
        // Pass the filter function as an argument to the AddUniverse method.
        _universe = AddUniverse(FundamentalFilterFunction);
    }

    private IEnumerable<Symbol> FundamentalFilterFunction(IEnumerable<Fundamental> fundamental)
    {
        var symbols = (from f in fundamental
                where f.HasFundamentalData
                select f.Symbol);

        if (LiveMode)
        {
            return symbols.Select(x => QuantConnect.Symbol.Create(x.Value, SecurityType.Cfd,
Market.InteractiveBrokers));
        }

        return symbols;
    }
}
```

Example

The simplest example of accessing the fundamental object would be harnessing the iconic PE ratio for a stock. This is a ratio of the price it commands to the earnings of a stock. The lower the PE ratio for a stock, the more affordable it appears.

```
// Select the top 50 by dollar volume and then select the top 10 by their PE ratio.
UniverseSettings.Asynchronous = true;
_universe = AddUniverse(
    fundamental =>
    {
        var symbols = (from f in fundamental
            where f.Price > 10 && f.HasFundamentalData && !Double.IsNaN(f.ValuationRatios.PERatio)
            orderby f.DollarVolume descending
            select f).Take(100)
            .OrderBy(f => f.ValuationRatios.PERatio).Take(10)
            .Select(f => f.Symbol);

        if (LiveMode)
        {
            return symbols.Select(x => QuantConnect.Symbol.Create(x.Value, SecurityType.Cfd,
Market.InteractiveBrokers));
        }

        return symbols;
    });
```

Practical Limitations

Fundamental universes allow you to select an unlimited universe of assets to analyze. Each asset in the universe consumes approximately 5MB of RAM, so you may quickly run out of memory if your universe filter selects many assets. If you backtest your algorithms in the Algorithm Lab, familiarize yourself with the RAM capacity of your backtesting and live trading nodes . To keep your algorithm fast and efficient, only subscribe to the assets you need.

Create Universes using ETF Constituents

To add an ETF Constituents universe, in the `Initialize` method, call the `Universe.ETF` method with a filter function, and pass it to the `AddUniverse` method. The filter function receives a list of `ETFConstituentUniverse` objects and must return a list of `Symbol` objects. The `Symbol` objects you return from the function are the constituents of the fundamental universe and LEAN automatically creates subscriptions for them. In live mode, call the `Symbol.Create` method to swap for a CFD version of the same `Symbol` . Don't call `AddCfd` in the filter function.

```
public class MyUniverseAlgorithm : QCAlgorithm {
    private Universe _universe;
    public override void Initialize()
    {
        UniverseSettings.Asynchronous = true;
        _universe = Universe.ETF("SPY", UniverseSettings, ETFConstituentsFilter);
        AddUniverse(_universe);
    }

    private IEnumerable<Symbol> ETFConstituentsFilter(IEnumerable<ETFConstituentUniverse> constituents)
    {
        // Get the 10 securities with the largest weight in the index
        var symbols = constituents.OrderByDescending(c => c.Weight).Take(10).Select(c => c.Symbol);
        if (LiveMode)
        {
            return symbols.Select(x => QuantConnect.Symbol.Create(x.Value, SecurityType.Cfd,
Market.InteractiveBrokers));
        }

        return symbols;
    }
}
```

Selection Frequency

Equity universes run on a daily basis by default. To adjust the selection schedule, see Schedule .

Live Trading Considerations

The live data for fundamental universe selection arrives at 6/7 AM Eastern Time (ET), so fundamental universe selection runs for live algorithms between 7 and 8 AM ET. This timing allows you to place trades before the market opens. Don't schedule anything for midnight because the universe selection data isn't ready yet.

Examples

The following examples demonstrate common practices for implementing Equity CFD's universe.

Example 1: Short Bottom-Weighted SPY Constituents

A subset of the SPY constituents outperforms the SPY, while many underperform the overall index. To exclude the ETF constituents that underperform the index, the following algorithm buys the SPY CFD and shorts the CFD of the 100 constituents in the index with the smallest weight in the ETF.

C#

```
public class EquityCfdUniverseAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private Dictionary<Symbol, decimal> _etfWeightBySymbol;
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(10000000);
        Settings.MinimumOrderMarginPortfolioPercentage = 0m;
        // Request SPY to trade. In live mode, request the IB CFD
        _spy = LiveMode ? AddCfd("SPY", market: Market.InteractiveBrokers).Symbol : AddEquity("SPY").Symbol;
        // The SPY ETF provides market hours for scheduled events.
        var spy = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
        // Refilter and rebalance weekly to avoid over-trading and high transaction costs.
        UniverseSettings.Schedule.On(DateRules.WeekEnd(spy));
        // Add a universe to select the CFD contracts of the SPY constituents.
        _universe = AddUniverse(Universe.ETF(spy, universeFilterFunc: SelectAssets));
        // Create a Scheduled Event to rebalance the portfolio.
        Schedule.On(
            DateRules.WeekStart(_spy),
            TimeRules.AfterMarketOpen(_spy, 1),
            Rebalance
        );
    }

    private IEnumerable<Symbol> SelectAssets(IEnumerable<ETFConstituentUniverse> constituents)
    {
        // Cache the constituent weights in a dictionary for filtering and position sizing.
        _etfWeightBySymbol = constituents
            .Where(c => c.Weight.HasValue)
            .ToDictionary(c => c.Symbol, c => c.Weight.Value);
        // Select the CFD contracts of the bottom-weighted 100 constituents.
        // They should have negative excess returns.
        return _etfWeightBySymbol
            .OrderBy(x => x.Value)
            .Take(100)
            .Select(x => !LiveMode ? x.Key : QuantConnect.Symbol.Create(x.Key.Value, SecurityType.Cfd,
Market.InteractiveBrokers));
    }

    private void Rebalance()
    {
        // Get the ETF weight of all the assets currently in the universe.
        var weightBySymbol = _universe.Selected.ToDictionary(symbol => symbol, symbol => _etfWeightBySymbol[symbol]);
        // Sell the CFD of the 100 ETF constituents with the lowest weight due to expected negative excess return.
        var targets = weightBySymbol.Select(x => new PortfolioTarget(x.Key, -x.Value)).ToList();
        // Buy the SPY CFD to eliminate systematic risk.
        targets.Add(new(_spy, 1.0m - weightBySymbol.Sum(kvp => kvp.Value)));
        // Place orders to rebalance the portfolio.
        SetHoldings(targets, liquidateExistingHoldings: true);
    }
}
```

Universes

Futures

Introduction

A Futures universe lets you select a basket of contracts for a single Future. LEAN models Future subscriptions as a universe of Future contracts. A Future universe is similar to an Option universe , except Future contracts don't have a strike price, so the universe filter primarily focuses on the contract expiration date.

Create Universes

To add a universe of Future contracts, in the `Initialize` method, call the `AddFuture` method. This method returns an `Future` object, which contains the continuous contract `Symbol` . The continuous contract `Symbol` is the key to access the contracts in the FutureChain that LEAN passes to the `OnData` method. The continuous contract `Mapped` property references the current contract in the continuous contract series. When you create the Future subscription, save a reference to the `Future` object so you can use it later in your algorithm.

C#

```
public class BasicFutureAlgorithm : QCAlgorithm
{
    private Future _future;
    public override void Initialize()
    {
        UniverseSettings.Asynchronous = true;
        _future = AddFuture(Futures.Currencies.BTC,
            extendedMarketHours: true,
            dataMappingMode: DataMappingMode.LastTradingDay,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            contractDepthOffset: 0);
        _future.SetFilter(0, 62);
    }

    public override void OnData(Slice data)
    {
        if (Portfolio.Invested)
        {
            return;
        }
        data.Bars.TryGetValue(_future.Symbol, out var continuousTradeBar);
        data.Bars.TryGetValue(_future.Mapped, out var mappedTradeBar);
        MarketOrder(_future.Mapped, 1);
    }

    // Track events when security changes its ticker, allowing the algorithm to adapt to these changes.
    public override void OnSymbolChangedEvents(SymbolChangedEvents symbolChangedEvents)
    {
        foreach (var (symbol, changedEvent) in symbolChangedEvents)
        {
            var oldSymbol = changedEvent.OldSymbol;
            var newSymbol = changedEvent.NewSymbol;
            var quantity = Portfolio[oldSymbol].Quantity;

            // Rolling over: to liquidate any position of the old mapped contract and switch to the newly mapped
contract
            var tag = $"Rollover - Symbol changed at {Time}: {oldSymbol} -> {newSymbol}";
            Liquidate(oldSymbol, tag: tag);
            if (quantity != 0) MarketOrder(newSymbol, quantity, tag: tag);
        }
    }
}
```

The following table describes the `AddFuture` method arguments:

Argument: `ticker`

The Future ticker. To view the supported assets in the US Futures dataset, see Supported Assets .

Data Type: `string` | Default Value: `None`

Argument: `resolution`

The resolution of the market data. To view the supported resolutions, see Resolutions . If you don't provide a value, it uses `Resolution.Minute` by default.

Data Type: `Resolution?` | Default Value: `null`

Argument: `market`

The Futures market. To view the supported markets in the US Futures dataset, see Supported Markets . If you don't provide a value, it uses the default Future market of your brokerage model .

Data Type: `string` | Default Value: `null`

Argument: `fillForward`

If true, the current slice contains the last available data even if there is no data at the current time.

Data Type: `bool` | Default Value: `true`

Argument: `leverage`

The leverage for this Future.

Data Type: `decimal` | Default Value: `Security.NullLeverage`

Argument: `extendedMarketHours`

If true, use data from the pre and post market sessions

Data Type: `bool` | Default Value: `false`

Argument: `dataMappingMode`

The contract mapping mode to use for the continuous future contract

Data Type: `DataMappingMode?` | Default Value: `null`

Argument: `dataNormalizationMode` The price scaling mode to use for the continuous future contract Data Type: `DataNormalizationMode?` | Default Value: `null`
Argument: `contractDepthOffset` The continuous future contract desired offset from the current front month. For example, 0 is the front month, 1 is the back month contract. Data Type: `int` | Default Value: `0`

Continous Contracts

By default, LEAN only subscribes to the continuous Future contract. A continuous Future contract represents a series of separate contracts stitched together to form a continuous price. If you need a lot of historical data to warm up an indicator, apply the indicator to the continuous contract price series. The `Future` object has a `Symbol` property and a `Mapped` property. The price of the `Symbol` property is the adjusted price of the continuous contract. The price of the `Mapped` property is the raw price of the currently selected contract in the continuous contract series.

```
C#

// Get the adjusted price of the continuous contract.
var adjustedPrice = Securities[_future.Symbol].Price;

// Get the raw price of the currently selected contract in the continuous contract series.
var rawPrice = Securities[_future.Mapped].Price;
```

The continuous Futures contract isn't a tradable security . You must place orders for a specific Futures contract. To access the currently selected contract in the continuous contract series, use the `Mapped` property of the `Future` object.

```
C#

// Place a market order for the currently selected contract in the continuous contract series.
MarkerOrder(_future.Mapped, 1);
```

To configure how LEAN identifies the current Future contract in the continuous series and how it forms the adjusted price between each contract, provide `dataMappingMode` , `dataNormalizationMode` , and `contractDepthOffset` arguments to the `AddFuture` method. The `Future` object that the `AddFuture` method returns contains a `Mapped` property that references the current contract in the continuous contract series. As the contracts roll over, the `Mapped` property references the next contract in the series and you receive a `SymbolChangedEvent` object in the `OnData` method. The `SymbolChangedEvent` references the old contract `Symbol` and the new contract `Symbol` . You can use `SymbolChangedEvents` to roll over contracts.

```
C#
public class BasicFutureAlgorithm : QCAlgorithm
{
    // Track when the continuous contract switches from one contract to the next.
    public override void OnSymbolChangedEvents(SymbolChangedEvents symbolChangedEvents)
    {
        foreach (var (symbol, changedEvent) in symbolChangedEvents)
        {
            var oldSymbol = changedEvent.OldSymbol;
            var newSymbol = changedEvent.NewSymbol;
            var quantity = Portfolio[oldSymbol].Quantity;

            // Rolling over: To liquidate the old mapped contract and switch to the new mapped contract.
            var tag = $"Rollover - Symbol changed at {Time}: {oldSymbol} -> {newSymbol}";
            Liquidate(oldSymbol, tag: tag);
            if (quantity != 0) MarketOrder(newSymbol, quantity, tag: tag);
        }
    }
}
```

In backtesting, the `SymbolChangedEvent` occurs at midnight Eastern Time (ET). In live trading, the live data for continuous contract mapping arrives at 6/7 AM ET, so that's when it occurs.

Data Normalization Modes

The `dataNormalizationMode` argument defines how the price series of two contracts are stitched together when the contract rollovers occur. The following `DataNormalizatoinMode` enumeration members are available for continuous contracts:

We use the entire Futures history to adjust historical prices. This process ensures you get the same adjusted prices, regardless of the backtest end date.

Data Mapping Modes

The `dataMappingMode` argument defines when contract rollovers occur. The `DataMappingMode` enumeration has the following members:

`Futures.Indices.VIX` (VX) doesn't support continous contract rolling with `DataMappingMode.OpenInterest` and `DataMappingMode.OpenInterestAnnuak` .

Contract Depth Offsets

The `contractDepthOffset` argument defines which contract to use. 0 is the front month contract, 1 is the following back month contract, and 2 is the second back month contract.

Filter Contracts

By default, LEAN doesn't add any contracts to the FuturesChain it passes to the `OnData` method.

To set a contract filter, in the `Initialize` method, call the `SetFilter` method of the `Future` object. The following table describes the available filter techniques:

`SetFilter(int minExpiry, int maxExpiry)` Selects the contracts that expire within the range you set. This filter runs asynchronously by default.
`SetFilter(Func<FutureFilterUniverse, FutureFilterUniverse> universeFunc)` Selects the contracts that a function selects.

```C#
// Select the contracts that expire within 182 days.
_future.SetFilter(0, 182);

// Select the front month contract.
_future.SetFilter(futureFilterUniverse => futureFilterUniverse.FrontMonth());
```

The following table describes the filter methods of the `FutureFilterUniverse` class:

`StandardsOnly()` Selects standard contracts
`FrontMonth()` Selects the front month contract
`BackMonths()` Selects the non-front month contracts
`BackMonth()` Selects the back month contracts
`Expiration(TimeSpan minExpiry, TimeSpan maxExpiry)` Selects contracts that expire within a range of dates relative to the current day
`Expiration(int minExpiryDays, int maxExpiryDays)` Selects contracts that expire within a range of dates relative to the current day
`Contracts(IEnumerable<Symbol> contracts)` Selects a list of contracts
`Contracts(Func<IEnumerable<Symbol>, IEnumerable< Symbol>> contractSelector)` Selects contracts that a selector function selects

The preceding methods return an `FutureFilterUniverse` , so you can chain the methods together.

```C#
// Select the front month standard contracts
_future.SetFilter(futureFilterUniverse => futureFilterUniverse.StandardsOnly().FrontMonth());
```

You can also define an isolated filter method.

```
// In Initialize
_future.SetFilter(Selector);

private FutureFilterUniverse Selector(FutureFilterUniverse futureFilterUniverse)
{
    return futureFilterUniverse.StandardsOnly().FrontMonth();
}
```

Navigate Futures Chains

`FuturesChain` objects represent an entire chain of contracts for a single underlying Future. They have the following properties:

To get the `FuturesChain` , index the `FuturesChains` property of the `Slice` with the continuous contract `Symbol` .

```
public override void OnData(Slice slice)
{
    if (slice.FuturesChains.TryGetValue(_symbol, out var chain))
    {
        // Example: Select the contract with the greatest open interest
        var contract = chain.OrderBy(x => x.OpenInterest).Last();
    }
}
```

You can also loop through the `FuturesChains` property to get each `FuturesChain` .

```
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.FuturesChains)
    {
        var continuousContractSymbol = kvp.Key;
        var chain = kvp.Value;
    }
}

public void OnData(FuturesChains futuresChains)
{
    foreach (var kvp in futuresChains)
    {
        var continuousContractSymbol = kvp.Key;
        var chain = kvp.Value;
    }
}
```

Historical Data

To get historical Future universe data, call the `History<FutureUniverse>` method. This method returns the all the available contracts for each trading day, not the subset of contracts that pass your universe filter.

```C#
public class FutureHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        var future = AddFuture(Futures.Indices.SP500EMini);
        var history = History<FutureUniverse>(future.Symbol, 5);
        foreach (var universe in history)
        {
            var endTime = universe.EndTime;
            foreach (FutureUniverse contract in universe)
            {
                var symbol = contract.Symbol;
                var expiry = contract.ID.Date;
                var price = contract.Close;
                var oi = contract.OpenInterest;
            }
        }
    }
}
```

The method represents each contract with a `FutureUniverse` object, which have the following properties:

Selection Frequency

By default, Futures universes run at the first time step of each day to select their contracts.

Examples

The following examples demonstrate some common Futures universes.

Example 1: Contracts Expiring On Quarter End

To reduce the need for rolling over, you can filter Future contracts that expire only on quarter ends to save transaction costs. The following example trades a simple EMA cross strategy on SP500 EMini Future contracts that expire on quarter ends.

C#

```
public class QuarterlyFuturesAlgorithm : QCAlgorithm
{
    private Future _future;
    private ExponentialMovingAverage _ema;
    // A variable to control the algorithm trading in daily frequency only, since the indicator signal is by daily.
    private int _day = -1;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(10000000);

        // Request ES Future data for trading. We only use the continuous contract for indicator values.
        _future = AddFuture(
            Futures.Indices.SP500EMini,
            extendedMarketHours: true,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            dataMappingMode: DataMappingMode.OpenInterest
        );
        // We only filter the ones expiring on quarter ends to reduce the transaction cost needed for rolling over
contracts.
        _future.SetFilter(futureFilterUniverse => futureFilterUniverse.ExpirationCycle(new int[] { 3, 6, 9, 12 }));

        // Create an EMA indicator to track trends for trade signal generation.
        _ema = EMA(_future.Symbol, 20, Resolution.Daily);
        // Warm up the indicator for immediate readiness.
        WarmUpIndicator(_future.Symbol, _ema, Resolution.Daily);

    }

    public override void OnData(Slice slice)
    {
        // Get the chain to trade from the filtered contracts.
        if (_day != slice.Time.Day && slice.FutureChains.TryGetValue(_future.Symbol, out var chain))
        {
            // Select the contract with the nearest expiry.
            var contract = chain.MinBy(x => x.Expiry);

            // Trade an EMA cross strategy to follow the trend.
            var quantity = Portfolio[_future.Symbol].Quantity;
            if (contract.LastPrice > _ema && quantity <= 0)
            {
                MarketOrder(contract.Symbol, 1-quantity);
            }
            else if (contract.LastPrice < _ema && quantity >= 0)
            {
                MarketOrder(contract.Symbol, -1-quantity);
            }

            _day = slice.Time.Day;
        }
    }
}
```

Example 2: Contracts Expiring Within One Week

The trading volume of Futures reaches its maximum during the last week of the trading period. The following algorithm selects contracts that expire within a week:

C#

```
public class WeeklyFuturesAlgorithm : QCAlgorithm
{
    private Future _future;
    private BollingerBands _bb;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add ES Futures for trading. The continuous contract data is
        // only used for indicator values.
        _future = AddFuture(
            Futures.Indices.SP500EMini,
            extendedMarketHours: true,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            dataMappingMode: DataMappingMode.OpenInterest
        );
        // Select the contract expiring within a week since they have
        // the highest liquidity.
        _future.SetFilter(futureFilterUniverse => futureFilterUniverse.Expiration(0, 7));
        // Create a Bollinger Band indicator to identify extreme prices
        // for trade signal generation.
        _bb = BB(_future.Symbol, 60, 2);
        // Warm up the indicator for so we can trade right away.
        WarmUpIndicator(_future.Symbol, _bb);
    }

    public override void OnData(Slice slice)
    {
        // Get the current Futures chain, if there is one.
        if (!slice.FutureChains.TryGetValue(_future.Symbol, out var chain))
        {
            return;
        }
        // Select the contract with the nearest expiry.
        var contract = chain.MinBy(x => x.Expiry);
        // Trade a mean-reversal strategy based on Bollinger Band signal.
        var holding = Portfolio[contract.Symbol];
        var price = contract.LastPrice;
        int targetQuantity;
        // Case 1: Short
        if (!holding.IsShort && price > _bb.UpperBand)
        {
            targetQuantity = -1;
        }
        // Case 2: Long
        else if (!holding.IsLong && price < _bb.LowerBand)
        {
            targetQuantity = 1;
        }
        // Case 3: Exit
        else if ((holding.IsShort && price < _bb.MiddleBand) || (holding.IsLong && price > _bb.MiddleBand))
        {
            targetQuantity = 0;
        }
        // Case 4: Hold
        else
        {
            return;
        }
        MarketOrder(contract.Symbol, targetQuantity - holding.Quantity);
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
EmaCrossFuturesFrontMonthAlgorithm.cs C#

Universes

Future Options

Introduction

A Future Option universe lets you select a basket of Option contracts on the contracts in a Futures universe .

Create Universes

To add a universe of Future Option contracts, in the `Initialize` method, define a Future universe and then pass the canonical `Symbol` to the `AddFutureOption` method.

```
C#
UniverseSettings.Asynchronous = true;
var future = AddFuture(Futures.Metals.Gold);
future.SetFilter(0, 90);
AddFutureOption(future.Symbol);
```

The following table describes the `AddFutureOption` method arguments:

Argument	Data Type	Description	Default Value
`symbol`	`Symbol`	The continuous Future contract Symbol. To view the supported assets in the US Future Options dataset, see Supported Assets .	
`optionFilter`	`Func<OptionFilterUniverse, OptionFilterUniverse>`	A function that selects Future Option contracts	`null`

To override the default pricing model of the Option, set a pricing model in a security initializer.

```C#
// In Initialize
var seeder = SecuritySeeder.Null;
SetSecurityInitializer(new MySecurityInitializer(BrokerageModel, seeder, this));

// Outside of the algorithm class
class MySecurityInitializer : BrokerageModelSecurityInitializer
{
    private QCAlgorithm _algorithm;

    public MySecurityInitializer(IBrokerageModel brokerageModel, ISecuritySeeder securitySeeder, QCAlgorithm
algorithm)
        : base(brokerageModel, securitySeeder)
    {
        _algorithm = algorithm;
    }

    public override void Initialize(Security security)
    {
        // First, call the superclass definition
        // This method sets the reality models of each security using the default reality models of the brokerage
model
        base.Initialize(security);

        // Next, set the price model
        if (security.Type == SecurityType.FutureOption) // Option type
        {
            security.PriceModel = OptionPriceModels.CrankNicolsonFD();
        }
    }
}
```

To override the initial guess of implied volatility , set and warm up the underlying volatility model .

Filter Contracts

By default, LEAN subscribes to the Option contracts that have the following characteristics:

- Standard type (weeklies and non-standard contracts are not available)
- Within 1 strike price of the underlying asset price
- Expire within 35 days

To adjust the universe of contracts, set a filter. The filter usually runs at every time step in your algorithm. When the filter selects a contract that isn't currently in your universe, LEAN adds the new contract data to the next `Slice` that it passes to the `OnData` method.

To set a contract filter, in the `Initialize` method, pass a filter function to the `AddFutureOption` method. The following table describes the available filter techniques:

```C#
AddFutureOption(future.Symbol, optionFilterUniverse => optionFilterUniverse.Strikes(-1, 1));
```

The following table describes the filter methods of the `OptionFilterUniverse` class:

`Strikes(int minStrike, int maxStrike)` Selects contracts that are within `minStrike` strikes below the underlying price and `maxStrike` strikes above the underlying price.

`CallsOnly()` Selects call contracts.
`PutsOnly()` Selects put contracts.
`StandardsOnly()` Selects standard contracts.
`IncludeWeeklys()` Selects non-standard weeklys contracts. This filter doesn't currently work. To add this functionality to our roadmap , react to this GitHub Issue with a thumbs up emoji.
`WeeklysOnly()` Selects weekly contracts. This filter doesn't currently work. To add this functionality to our roadmap , react to this GitHub Issue with a thumbs up emoji.
`FrontMonth()` Selects the front month contract.
`BackMonths()` Selects the non-front month contracts.
`BackMonth()` Selects the back month contracts.
`Expiration(int minExpiryDays, int maxExpiryDays)` Selects contracts that expire within a range of dates relative to the current day.
`Contracts(IEnumerable<Symbol> contracts)` Selects a list of contracts.
`Contracts(Func<IEnumerable<Symbol>, IEnumerable< Symbol>> contractSelector)` Selects contracts that a selector function selects.

The preceding methods return an `OptionFilterUniverse` , so you can chain the methods together.

```
AddFutureOption(future.Symbol, optionFilterUniverse => optionFilterUniverse.Strikes(-1, 1).CallsOnly());
```

To perform thorough filtering on the `OptionFilterUniverse`, define an isolated filter method.

```
# In Initialize
AddFutureOption(future.Symbol, Selector);

private OptionFilterUniverse Selector(OptionFilterUniverse optionFilterUniverse)
{
    var symbols = optionFilterUniverse.PutsOnly();
    var strike = symbols.Select(symbol => symbol.ID.StrikePrice).Min();
    symbols = symbols.Where(symbol => symbol.ID.StrikePrice == strike);
    return optionFilterUniverse.Contracts(symbols);
}
```

Some of the preceding filter methods only set an internal enumeration in the `OptionFilterUniverse` that it uses later on in the filter process. This subset of filter methods don't immediately reduce the number of contract `Symbol` objects in the `OptionFilterUniverse`.

Navigate Option Chains

`OptionChain` objects represent an entire chain of Option contracts for a single underlying security.

To get the `OptionChain`, loop through the `OptionChains` property. After you get the `OptionChain`, you can sort and filter the Option contracts in the chain.

```
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.OptionChains)
    {
        var optionChain = kvp.Value;
        // Example: Find 5 put contracts that are closest to at-the-money (ATM) and have the farthest expiration
        var contracts = optionChain
            .Where(x => x.Right == OptionRight.Put)
            .OrderByDescending(x => x.Expiry)
            .ThenBy(x => Math.Abs(chain.Underlying.Price - x.Strike))
            .Take(5);

        // Select the contract with the delta closest to -0.5
        var contract = contracts.OrderBy(x => Math.Abs(-0.5m - x.Greeks.Delta)).FirstOrDefault();
    }
}
```

You can also iterate through the `FuturesChains` first.

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.FuturesChains)
    {
        var continuousContractSymbol = kvp.Key;
        var futuresChain = kvp.Value;

        // Select a Future Contract and create its canonical FOP Symbol
        var futuresContract = futuresChain.First();
        var canonicalFOPSymbol = QuantConnect.Symbol.CreateCanonicalOption(futuresContract.Symbol);
        if (slice.OptionChains.TryGetValue(canonicalFOPSymbol, out var fopChain))
        {
            foreach (var contract in fopChain)
            {
                //
            }
        }
    }
}
```

`OptionChain` objects have the following properties:

Historical Data

To get historical Future Option universe data, call the `History<FutureUniverse>` method with the canoncial Option `Symbol`. This method returns the all the available contracts for each trading day, not the subset of contracts that pass your universe filter.

```C#
public class FutureOptionHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Select a Futures contract.
        var future = AddFuture(Futures.Indices.SP500EMini);
        var futuresContract = FuturesChain(future.Symbol).First();
        // Create the canoncial Option Symbol for this Futures contract.
        var canonical = QuantConnect.Symbol.CreateCanonicalOption(futuresContract.Symbol);
        // Get the historical Option universe data for this Futures contract.
        var history = History<OptionUniverse>(canonical, 5);
        foreach (var universe in history)
        {
            var endTime = universe.EndTime;
            foreach (OptionUniverse contract in universe)
            {
                var symbol = contract.Symbol;
                var expiry = contract.ID.Date;
                var price = contract.Close;
            }
        }
    }
}
```

The method represents each contract with a `OptionUniverse` object, which have the following properties:

Selection Frequency

By default, Future Option universes run at the first time step of each day to select their contracts.

Examples

The following examples demonstrate some common Future Option universes.

Example 1: 0DTE Contracts

0DTE Options are Option contracts that expire on the same day you trade them. The following algorithm selects 0DTE Future

Option contracts for the E-mini S&P 500 that fall within 3 strikes of the underlying price.

```C#
public class ZeroDTEFutureOptionUniverseAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        var future = AddFuture(Futures.Indices.SP500EMini);
        future.SetFilter(0, 90);
        AddFutureOption(
            future.Symbol,
            u => u.IncludeWeeklys().Expiration(0, 0).Strikes(-3, 3)
        );
    }
}
```

For full trading examples of Future Options, see Examples .

Universes

Index Options

Introduction

An Index Option universe lets you select a basket of Option contracts on an Index.

Create Universes

To add a universe of Index Option contracts, in the `Initialize` method, call the `AddIndexOption` method. This method returns an `Option` object, which contains the canonical `Symbol` . You can't trade with the canonical Option `Symbol` , but save a reference to it so you can easily access the Option contracts in the OptionChain that LEAN passes to the `OnData` method. The method to create the universe depends on if the Index Options you want require a target ticker.

Create Non-Standard Universes

Non-standard Option contracts are popular among Option traders. To create a universe of non-standard Index Options like weekly VIX contracts, pass the index Symbol and target Option ticker to the `AddIndexOption` method.

C#

```
public class WeeklyIndexOptionAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    public override void Initialize()
    {
        var underlying = AddIndex("VIX").Symbol;
        var option = AddIndexOption(underlying, "VIXW");
        option.SetFilter(universe => universe.IncludeWeeklys().Expiration(0, 7).Delta(0.35m, 0.75m));
        _symbol = option.Symbol;
    }

    public override void OnData(Slice data)
    {
        if (data.OptionChains.TryGetValue(_symbol, out var chain))
        {
            var contract = chain.OrderBy(x => x.Expiry).ThenBy(x => x.Greeks.Delta).FirstOrDefault();
        }
    }
}
```

The following table describes the `AddIndexOption` method arguments for non-standard universes:

Argument	Data Type	Description	Default Value
`underlying`	`Symbol`	The underlying Index `Symbol` . To view the supported indices, see Supported Assets .	
`targetOption`	`string`	The target Option ticker. To view the supported target Options, see Supported Assets .	
`resolution`	`Resolution?`	The resolution of the market data. To view the supported resolutions, see Resolutions . The Index resolution must be less than or equal to the Index Option resolution. For example, if you set the Index resolution to minute, then you must set the Index Option resolution to minute, hour, or daily.	`null`
`market`	`string`	The Index Option market.	`Market.USA`
`fillForward`	`bool`	If true, the current slice contains the last available data even if there is no data at the current time.	`true`

If you add an Option universe for an underlying Index that you don't have a subscription for, LEAN automatically subscribes to the underlying Index.

Create Standard Universes

To create a universe of Index Options based on an index like VIX, SPX, or NDX, pass the index ticker to the `AddIndexOption` method.

```
C#
var option = AddIndexOption("SPX");
option.SetFilter(universe => universe.Expiration(0, 60).Delta(0.35m, 0.75m););
 _symbol = option.Symbol;
```

The following table describes the `AddIndexOption` method arguments for standard universes:

Argument	Data Type	Description	Default Value
`ticker`	`string`	The underlying Index ticker. To view the supported indices, see Supported Assets .	
`resolution`	`Resolution?`	The resolution of the market data. To view the supported resolutions, see Resolutions .	`null`
`market`	`string`	The Index Option market.	`Market.USA`
`fillForward`	`bool`	If true, the current slice contains the last available data even if there is no data at the current time.	`true`

If you add an Option universe for an underlying Index that you don't have a subscription for, LEAN automatically subscribes to the underlying Index and sets its fill forward property to match that of the Index Option universe.

Filter by Investment Strategy

Options trading strategies consist of simultaneously buying or selling one or more Option contracts on the same underlying Index, but with different features.

```
C#
// Example 1: Select a Straddle
option.SetFilter(optionFilterUniverse => optionFilterUniverse.Straddle(30, 5, 10));

// Example 2: Select the contracts (including weeklys) that expire in the next 30 days that form an Iron Condor
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().Strikes(-20, 20).Expiration(0,
30).IronCondor(30, 5, 10));

// Example 3: Select the 0DTE contracts that form an Strangle
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().Expiration(0, 0).Strangle(30, 5, -
10));
```

The following table describes the filter methods of the `OptionFilterUniverse` class that select contracts for Option strategies:

`NakedCall(int minDaysTillExpiry, decimal strikeFromAtm)` Selects a call contract to form Naked Call , Covered Call , or Protective Call Option strategies.
`NakedPut(int minDaysTillExpiry, decimal strikeFromAtm)` Selects a put contract to form Naked Put , Covered Put , or Protective Put Option strategies.
`CallSpread(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects two call contracts to form Bull Call Spread or Bear Call Spread Option strategies.
`PutSpread(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects two put contracts to form Bull Put Spread or Bear Put Spread Option strategies.

`CallCalendarSpread(decimal strikeFromAtm, int minNearDaysTillExpiry, int minFarDaysTillExpiry)` Selects two call contracts to form Long Call Calendar Spread or Short Call Calendar Spread Option strategies.
`PutCalendarSpread(decimal strikeFromAtm, int minNearDaysTillExpiry, int minFarDaysTillExpiry)` Selects two put contracts to form Long Put Calendar Spread or Short Put Calendar Spread Option strategies.
`Strangle(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects two contracts to form Long Strangle or Short Strangle Option strategies.
`Straddle(int minDaysTillExpiry)` Selects two contracts to form Long Straddle or Short Straddle Option strategies.
`ProtectiveCollar(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects two contracts to form Protective Collar Option strategies.
`Conversion(int minDaysTillExpiry, decimal strikeFromAtm)` Selects two contracts to form Conversion or Reverse Conversion Option strategies.
`CallButterfly(int minDaysTillExpiry, decimal strikeSpread)` Selects three contracts to form Long Call Butterfly or Short Call Butterfly Option strategies.
`PutButterfly(int minDaysTillExpiry, decimal strikeSpread)` Selects three contracts to form Long Put Butterfly or Short Put Butterfly Option strategies.
`IronButterfly(int minDaysTillExpiry, decimal strikeSpread)` Selects four contracts to form Long Iron Butterfly or Short Iron Butterfly Option strategies.
`IronCondor(int minDaysTillExpiry, decimal nearStrikeSpread, decimal farStrikeSpread)` Selects four contracts to form Long Iron Condor or Short Iron Condor Option strategies.
`BoxSpread(int minDaysTillExpiry, decimal strikeSpread)` Selects four contracts to form Box Spread or Short Box Spread Option strategies.
`JellyRoll(decimal strikeFromAtm, int minNearDaysTillExpiry, int minFarDaysTillExpiry)` Selects four contracts to form Jelly Roll or Short Jelly Roll Option strategies.

`CallLadder(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal middleStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects four contracts to form Bear Call Ladder or Bull Call Ladder Option strategies.
`PutLadder(int minDaysTillExpiry, decimal higherStrikeFromAtm, decimal middleStrikeFromAtm, decimal lowerStrikeFromAtm)` Selects four contracts to form Bear Put Ladder or Bull Put Ladder Option strategies.

The preceding methods return an `OptionFilterUniverse`, so you can chain the methods together.

Filter by Implied Volatility and Greeks

Option price models compute the theoretical price of Option contracts, their implied volatility, and their Greek values. Theoretical prices can help you detect undervalued and overvalued contracts, implied volatility can provide you insight into the upcoming volatility of the underlying security, and Greek values can help you hedge your portfolio.

```C#
// Example 1: Select the contracts with delta between 0.25 and 0.75
option.SetFilter(optionFilterUniverse => optionFilterUniverse.Delta(0.25m, 0.75m));

// Example 2: Select the contracts (including weeklys) that expire in the next 90 days with implied volatility below 20%
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().IV(0, 20).Expiration(0, 90));

// Example 3: Select the contracts (including weeklys) that expire in the next 30 days with implied volatility below 20% that forms an Iron Condor
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().IV(0, 20).Expiration(0, 30).IronCondor(30, 5, 10));
```

The following table describes the filter methods of the `OptionFilterUniverse` class that select contract for a range of implied volatility, Greek values, and open interest:

`IV(decimal min, decimal max)`
`ImpliedVolatility(decimal min, decimal max)`
 	
Selects a contract with implied volatility within the range you set.

`D(decimal min, decimal max)`
`Delta(decimal min, decimal max)`
 	
Selects a contract with delta within the range you set.

`G(decimal min, decimal max)`
`Gamma(decimal min, decimal max)`
 	
Selects a contract with gamma within the range you set.

`R(decimal min, decimal max)`
`Rho(decimal min, decimal max)`
 	
Selects a contract with rho within the range you set.

`V(decimal min, decimal max)`
`Vega(decimal min, decimal max)`
 	
Selects a contract with vega within the range you set.

`T(decimal min, decimal max)`
`Theta(decimal min, decimal max)`
 	
Selects a contract with theta within the range you set.

`OI(long min, long max)`
`OpenInterest(decimal min, decimal max)`
 	
Selects a contract with open interest within the range you set.

The preceding methods return an `OptionFilterUniverse`, so you can chain the methods together.

To perform thorough filtering on the `OptionFilterUniverse`, define an isolated filter method.

```
C#
```

```
// Select the put contracts with the lowest strike price.
option.SetFilter(Selector);

private OptionFilterUniverse Selector(OptionFilterUniverse optionFilterUniverse)
{
    return universe
        .Delta(0.5m, 1.5m)
        .Gamma(0.0001m, 0.0006m)
        .Vega(0.01m, 1.5m)
        .Theta(-2.0m, -0.5m)
        .Rho(0.5m, 3.0m)
        .ImpliedVolatility(1.0m, 3.0m)
        .OpenInterest(100, 500);
}
```

Filter by Other Contract Properties

To set a contract filter, in the `Initialize` method, call the `SetFilter` method of the `Option` object.

```
C#
// Select contracts that have a strike price within 1 strike level above and below the underlying price
option.SetFilter(minStrike: -1, maxStrike: 1);

// Select contracts that expire within 30 days
option.SetFilter(minExpiry: TimeSpan.FromDays(0), maxExpiry: TimeSpan.FromDays(30));

// Select contracts that have a strike price within 1 strike level and expire within 30 days
option.SetFilter(minStrike: -1, maxStrike: 1, minExpiry: TimeSpan.FromDays(0), maxExpiry: TimeSpan.FromDays(30));

// Select call contracts
option.SetFilter(optionFilterUniverse => optionFilterUniverse.CallsOnly());
```

The following table describes the available filter techniques:

`SetFilter(int minStrike, int maxStrike)` Selects the contracts that have a strike price within a minimum and maximum strike level relative to the underlying price. For example, say the underlying price is 302*andthereare*strikes*atevery*5. If you set `minStrike` to -1 and `maxStrike` to 1, LEAN selects the contracts that have a strike of 300*or*305. This filter runs asynchronously by default.
`SetFilter(TimeSpan minExpiry, TimeSpan maxExpiry)` Selects the contracts that expire within the range you set. This filter runs asynchronously by default.
`SetFilter(int minStrike, int maxStrike, TimeSpan minExpiry, TimeSpan maxExpiry)` Selects the contracts that expire and have a strike within the range you set. This filter runs asynchronously by default.
`SetFilter(Func<OptionFilterUniverse, OptionFilterUniverse> universeFunc)` Selects the contracts that a function selects.

The following table describes the filter methods of the `OptionFilterUniverse` class:

`Strikes(int minStrike, int maxStrike)` Selects contracts that are within `minStrike` strikes below the underlying price and `maxStrike` strikes above the underlying price.
`CallsOnly()` Selects call contracts.
`PutsOnly()` Selects put contracts.

`StandardsOnly()` Selects standard contracts.
`IncludeWeeklys()` Selects non-standard weeklys contracts.
`WeeklysOnly()` Selects weekly contracts.
`FrontMonth()` Selects the front month contract.
`BackMonths()` Selects the non-front month contracts.
`BackMonth()` Selects the back month contracts.
`Expiration(int minExpiryDays, int maxExpiryDays)` Selects contracts that expire within a range of dates relative to the current day.
`Contracts(IEnumerable<Symbol> contracts)` Selects a list of contracts.
`Contracts(Func<IEnumerable<Symbol>, IEnumerable< Symbol>> contractSelector)` Selects contracts that a selector function selects.

The preceding methods return an `OptionFilterUniverse` , so you can chain the methods together.

```C#
// Example 1: Select the front month call contracts
option.SetFilter(optionFilterUniverse => optionFilterUniverse.CallsOnly().FrontMonth());

// Example 2: Select the contracts (including weeklys) that expire in the next 90 days
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().Strikes(-20, 20).Expiration(0, 90));

// Example 3: Select the contracts (including weeklys) that expire in the next 30 days that form an Iron Condor
option.SetFilter(optionFilterUniverse => optionFilterUniverse.IncludeWeeklys().Strikes(-20, 20).Expiration(0,
30).IronCondor(30, 5, 10));
```

Some of the preceding filter methods only set an internal enumeration in the `OptionFilterUniverse` that it uses later on in the filter process. This subset of filter methods don't immediately reduce the number of contract `Symbol` objects in the `OptionFilterUniverse` .

Default Filter

By default, LEAN subscribes to the Option contracts that have the following characteristics:

- Standard type (exclude weeklys)
- Within 1 strike price of the underlying asset price
- Expire within 35 days

To adjust the universe of contracts, set a filter. The filter usually runs at the first bar of every day. When the filter selects a contract that isn't currently in your universe, LEAN adds the new contract data to the next `Slice` that it passes to the `OnData` method.

Navigate Intraday Option Chains

`OptionChain` objects represent an entire chain of Option contracts for a single underlying security.

To get the `OptionChain` , index the `OptionChains` property of the `Slice` with the canonical `Symbol` . After you get the `OptionChain` , you can sort and filter the Option contracts in the chain.

```
C#
public override void OnData(Slice slice)
{
    // Try to get the OptionChain using the canonical symbol
    if (slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        // Example: Find 5 put contracts that are closest to at-the-money (ATM) and have the farthest expiration
        var contracts = chain
            .Where(x => x.Right == OptionRight.Put)
            .OrderByDescending(x => x.Expiry)
            .ThenBy(x => Math.Abs(chain.Underlying.Price - x.Strike))
            .Take(5);

        // Select the contract with the delta closest to -0.5
        var contract = contracts.OrderBy(x => Math.Abs(-0.5m - x.Greeks.Delta)).FirstOrDefault();
    }
}
```

You can also loop through the `OptionChains` property to get each `OptionChain` .

```
C#
public override void OnData(Slice slice)
{
    // Iterate all received Canonical Symbol-OptionChain key-value pairs
    foreach (var kvp in slice.OptionChains)
    {
        var canoncialSymbol = kvp.Key;
        var chain = kvp.Value;
        var contracts = chain.Contracts;
    }
}
```

`OptionChain` objects have the following properties:

Navigate Daily Option Chains

To get the daily, pre-calculated Greeks and implied volaility of all the currently tradable contracts, call the `OptionChain` method. This method returns a collection of `OptionUniverse` objects, where each object represents a contract. The data this method returns contains information on all the currently tradable contracts, not just the contracts that pass your filter.

```C#
// In the initialize method, create a Scheduled Event to get the Option
// chain and rebalance the portfolio.
Schedule.On(DateRules.WeekStart(_symbol), TimeRules.AfterMarketOpen(_symbol, 1), Rebalance);

public void Rebalance()
{
    foreach (var optionUniverse in OptionChain(_symbol))
    {
        var close = optionUniverse.Close;
        var oi = optionUniverse.OpenInterest;
        var delta = optionUniverse.Greeks.Delta;
    }
}
```

The `OptionChain` method was previously known as the `OptionChainProvider.GetOptionContractList` method.

`OptionUniverse` objects have the following properties:

Greeks and Implied Volatility

There are several ways to get the implied volatility and Greeks of contracts.

The Greeks and IV values in the filter function are the daily, pre-calculated values based on the end of the previous trading day.

```C#
public override void Initialize()
{
    var option = AddIndexOption("SPX");
    option.SetFilter(universe => universe.IncludeWeeklys().Delta(0.3m, 0.7m).Expiration(0, 7));
}
```

To calculate the values, we use our implementation of the Black-Scholes pricing model, which accounts for the interest rate and dividend payments when applicable. For example, VIX doesn't have dividends. We use SPY for SPX and QQQ for NDX.

You can't customize the Greeks and IV values that the filter function receives. However, you can create indicators to customize how the Greeks and IV are calculated for the contracts already in your universe.

To override the default pricing model of the Option, set a pricing model . If you set a price model, it sets the price model of the individual contracts in the universe.

```C#
// Set PriceModel property to use the Crank-Nicolson finite-difference model to price the Options.
option.PriceModel = OptionPriceModels.CrankNicolsonFD();
```

To override the initial guess of implied volatility , set and warm up the underlying volatility model .

Historical Data

To get historical chains for an Index Option, call the `History<OptionUniverse>` method with the canonical Option `Symbol` . This method returns the entire Option chain for each trading day, not the subset of contracts that pass your universe filter.

```
C#
public class OptionHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        var option = AddIndexOption("SPX");
        var history = History<OptionUniverse>(option.Symbol, 5);
        foreach (var chain in history)
        {
            var endTime = chain.EndTime;
            var filteredContracts = chain.Data
                .Select(contract => contract as OptionUniverse)
                .Where(contract => contract.Greeks.Delta > 0.3m);
            foreach (var contract in filteredContracts)
            {
                var price = contract.Price;
                var iv = contract.ImpliedVolatility;
            }
        }
    }
}
```

The Greeks and IV values that you get from a history request of the Option universe are the daily, pre-calculated values based on the end of the previous trading day. To get the intraday values or to customize the Greeks and IV calculations, create some Option indicators .

The `History<OptionUniverse>` method represents each contract with an `OptionUniverse` object, which have the following properties:

Selection Frequency

By default, Index Option universes run at the first time step of each day to select their contracts.

Examples

The following examples demonstrate some common Index Option universes.

Example 1: 0DTE Contracts

0DTE Options are Option contracts that expire on the same day you trade them. The following algorithm selects 0DTE Option contracts for the SPX Index that fall within 3 strikes of the underlying price.

```
C#
public class ZeroDTEIndexOptionUniverseAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        var index = AddIndex("SPX");
        var indexOption = AddIndexOption(index.Symbol, "SPXW");
        indexOption.SetFilter(u => u.IncludeWeeklys().Expiration(0, 0).Strikes(-3, 3));
    }
}
```

For trading full examples of Index Option Universe, please refer to the Securities > Index Options > Requesting Data > Universes page.

Universes

Crypto

Introduction

A Crypto universe lets you select a basket of Cryptocurrencies based on `CryptoUniverse` data.

Crypto Universes

To add a universe of Cryptocurrencies, in the `Initialize` method, pass a `CryptoUniverse` to the `AddUniverse` method.

```C#
private Universe _universe;
public override void Initialize()
{
    UniverseSettings.Asynchronous = true;
    UniverseSettings.Resolution = Resolution.Daily;
    SetBrokerageModel(BrokerageName.Coinbase, AccountType.Cash);

    // Add crypto universe selection
    _universe = AddUniverse(CryptoUniverse.Coinbase(universeDay => from x in universeDay select x.Symbol));
}
```

The following table shows the helper methods to create Crypto universes for the supported exchanges:

Brokerage Name	Helper Method	Example
`BrokerageName.Binance`	`CryptoUniverse.Binance`	Example
`BrokerageName.BinanceUS`	`CryptoUniverse.BinanceUS`	Example
`BrokerageName.Bitfinex`	`CryptoUniverse.Bitfinex`	Example
`BrokerageName.Bybit`	`CryptoUniverse.Bybit`	Example
`BrokerageName.Coinbase`	`CryptoUniverse.Coinbase`	Example
`BrokerageName.Kraken`	`CryptoUniverse.Kraken`	Example

The following table describes the `CryptoUniverse` method arguments:

Argument	Data Type	Description	Default Value
`selector`	`Func<IEnumerable<CryptoUniverse>, IEnumerable<Symbol>>`	A function to select some of the Cryptocurrencies for the universe.	
`universeSettings`	`UniverseSettings`	The universe settings .	`null`

The filter function receives `CryptoUniverse` objects, which represent one of the Cryptocurrencies in the market. The `Symbol` objects that the filter function returns represent the universe constituents. `CryptoUniverse` objects have the following attributes:

To perform thorough filtering on the `CryptoUniverse` objects, define an isolated filter method.

```
C#
private Universe _universe;
public override void Initialize()
{
    UniverseSettings.Asynchronous = true;
    UniverseSettings.Resolution = Resolution.Daily;
    SetBrokerageModel(BrokerageName.Coinbase, AccountType.Cash);

    // Add crypto universe selection
    _universe = AddUniverse(CryptoUniverse.Coinbase(UniverseFilter));
}

private IEnumerable<Symbol> UniverseFilter(IEnumerable< CryptoUniverse> universeDay)
{
    return universeDay.Where(cf => cf.Volume >= 100m && cf.VolumeInUsd > 10000m).Select(x => x.Symbol);
}
```

Historical Data

To get historical Cryptocurrency universe data, call the `History` method with the `Universe` object and the lookback period. The return type is a `IEnumerable<BaseDataCollection>` and you have to cast its items to `CryptoUniverse` .

```
C#
// Get 30 days of history for the Crypto universe.
var history = History(_universe, 30, Resolution.Daily);
foreach (var universeDay in history)
{
    foreach (CryptoUniverse universeItem in universeDay)
    {
        Log($"{universeItem.Symbol} price at {universeItem.EndTime}: {universeItem.Close}");
    }
}
```

For more information about Cryptocurrency data, see Crypto .

Alternative Data Universes

An alternative data universe lets you select a basket of Cryptocurrencies based on an alternative dataset that's linked to them. If you use an alternative data universe, you limit your universe to only the securities in the dataset, which avoids unnecessary subscriptions. Currently, only the Crypto Market Cap alternative dataset supports universe selection for Crypto.

Selection Frequency

Crypto universes run on a daily basis by default. To adjust the selection schedule, see Schedule .

Live Trading Considerations

In live mode, the pipeline has a 16-hour delay. Your algorithm receives the `CryptoUniverse` objects at around 16:00-16:30 Coordinated Universal Time (UTC) each day, depending on the data processing task.

Examples

The following examples demonstrate some common Crypto universes.

Example 1: Highly Liquid Crypto Universe

To fairly compare the liquidity between Crypto pairs, use the `VolumeInUsd` property of the `CryptoUniverse` objects during universe selection. The following algorithm selects the 10 Crypto pairs with the greatest USD volume on Bitfinex:

```C#
public class HighlyLiquidCryptoUniverseAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        AddUniverse(
            CryptoUniverse.Bitfinex(data => data
                .Where(x => x.VolumeInUsd != null)
                .OrderByDescending(x => x.VolumeInUsd)
                .Take(10)
                .Select(x => x.Symbol)
            )
        );
    }
}
```

Example 2: Highly Volatile Crypto Universe

The following algorithm creates a universe of volatile Crypto pairs on Binance by selecting the pairs with the largest trading range in the previous day:

```C#
public class HighlyVolatileCryptoUniverseAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        AddUniverse(
            CryptoUniverse.Binance(data => data
                .OrderByDescending(x => (x.High - x.Low) / x.Low)
                .Take(10)
                .Select(x => x.Symbol)
            )
        );
    }
}
```

Example 3: Large Cap Crypto Universe

The following algorithm uses the CoinGecko Crypto Market Cap dataset to create a universe of the largest Crypto pairs on Coinbase. To avoid trading errors, it only selects pairs that have a quote currency that matches your account currency . At noon each day, it rebalances the portfolio to give equal weight to each coin in the universe.

C#

```
public class LargeCapCryptoUniverseAlgorithm : QCAlgorithm
{
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Set the account currency. USD is already the default value. Change it here if you want.
        SetAccountCurrency("USD");
        // Get the pairs that our brokerage supports and have a quote currency that
        // matches your account currency. We need this list in the universe selection function.
        var market = Market.Coinbase;
        var marketPairs = SymbolPropertiesDatabase.GetSymbolPropertiesList(market)
            .Where(x => x.Value.QuoteCurrency == AccountCurrency)
            .Select(x => x.Key.Symbol)
            .ToList();
        // Add the Crypto universe and define the selection function.
        _universe = AddUniverse<CoinGeckoUniverse>(
            data => data
                .Select(c => c as CoinGecko)
                // Select the coins that the brokerage supports and have a quote currency that
                // matches our account currency.
                .Where(c => marketPairs.Contains(c.Coin + AccountCurrency))
                // Select the 10 largest coins.
                .OrderByDescending(c => c.MarketCap)
                .Take(10)
                // Create the Symbol objects of the selected coins.
                .Select(c => c.CreateSymbol(market, AccountCurrency)
            )
        );
        // Add a Sheduled Event to rebalance the portfolio.
        Schedule.On(DateRules.EveryDay(), TimeRules.At(12, 0), Rebalance);
    }

    private void Rebalance()
    {
        if (_universe.Selected == null || _universe.Selected.Count == 0)
        {
            return;
        }
        var symbols = _universe.Selected.Where(symbol => Securities[symbol].Price != 0).ToList();
        // Liquidate coins that are no longer in the universe.
        var targets = Portfolio.Values
            .Where(holding => holding.Invested && !symbols.Contains(holding.Symbol))
            .Select(holding => new PortfolioTarget(holding.Symbol, 0))
            .ToList();
        // Form an equal weighted portfolio of the coins in the universe.
        targets.AddRange(symbols.Select(symbol => new PortfolioTarget(symbol, 0.5m / symbols.Count)));
        // Place orders to rebalance the portfolio.
        SetHoldings(targets);
    }
}
```

Universes

Custom Universes

Introduction

A custom universe lets you select a basket of assets from a custom dataset.

Data Sources

You can gather your custom data from any of the following sources:

The data source should serve data in chronological order and each data point should have a unique timestamp. Each request has a 1 second overhead, so bundle samples together for fast execution.

Define Custom Universe Types

Custom universes should extend the `BaseData` class. Extensions of the `BaseData` class must implement a `GetSource` and `Reader` method.

The `GetSource` method in your custom data class instructs LEAN where to find the data. This method must return a `SubscriptionDataSource` object, which contains the data location and format (`SubscriptionTransportMedium`). You can even change source locations for backtesting and live modes. We support many different data sources.

The `Reader` method of your custom data class takes one line of data from the source location and parses it into one of your custom objects. You can add as many properties to your custom data objects as you need, but must set `Symbol` and `EndTime` properties. When there is no useable data in a line, the method should return `null` . LEAN repeatedly calls the `Reader` method until the date/time advances or it reaches the end of the file.

```C#
//Example custom universe data; it is virtually identical to other custom data types.
public class MyCustomUniverseDataClass : BaseData
{
    public int CustomAttribute1;
    public decimal CustomAttribute2;
    public override DateTime EndTime
    {
        // define end time as exactly 1 day after Time
        get { return Time + QuantConnect.Time.OneDay; }
        set { Time = value - QuantConnect.Time.OneDay; }
    }

    public override SubscriptionDataSource GetSource(SubscriptionDataConfig config, DateTime date, bool isLiveMode)
    {
        return new SubscriptionDataSource(@"your-remote-universe-data", SubscriptionTransportMedium.RemoteFile);
    }

    public override BaseData Reader(SubscriptionDataConfig config, string line, DateTime date, bool isLiveMode)
    {
        var items = line.Split(",");

        // Generate required data, then return an instance of your class.
        return new MyCustomUniverseDataClass
        {
            EndTime = Parse.DateTimeExact(items[0], "yyyy-MM-dd"),
            Symbol = Symbol.Create(items[1], SecurityType.Equity, Market.USA),
            CustomAttribute1 = int.Parse(items[2]),
            CustomAttribute2 = decimal.Parse(items[3], NumberStyles.Any, CultureInfo.InvariantCulture)
        };
    }
}
```

Your `Reader` method should return objects in chronological order. If an object has a timestamp that is the same or earlier than the timestamp of the previous object, LEAN ignores it.

If you need to create multiple objects in your `Reader` method from a single `line` , follow these steps:

1. In the `GetSource` method, pass `FileFormat.UnfoldingCollection` as the third argument to the `SubscriptionDataSource` constructor.
2. In the `Reader` method, order the objects by their timestamp and then return a `BaseDataCollection(endTime, config.Symbol, objects)` where `objects` is a list of your custom data objects.

```C#
public class MyCustomUniverseDataClass : BaseData
{
    public int CustomAttribute1 { get; set; }
    public string Ticker { get; set; }
    public DateTime Date { get; set; }

    public override DateTime EndTime
    {
        // define end time as exactly 1 day after Time
        get { return Time + QuantConnect.Time.OneDay; }
        set { Time = value - QuantConnect.Time.OneDay; }
    }

    public MyCustomUniverseDataClass()
    {
        Symbol = Symbol.Empty;
        DataType = MarketDataType.Base;
    }

    public override SubscriptionDataSource GetSource(SubscriptionDataConfig config, DateTime date, bool isLiveMode)
    {
        return new SubscriptionDataSource(@"your-data-source-url",
            SubscriptionTransportMedium.RemoteFile,
            FileFormat.UnfoldingCollection);
    }

    public override BaseData Reader(SubscriptionDataConfig config, string line, DateTime date, bool isLiveMode)
    {
        var items = JsonConvert.DeserializeObject<List<MyCustomUniverseDataClass>>(line);
        var endTime = items.Last().Date;

        foreach (var item in items)
        {
            item.Symbol = Symbol.Create(item.Ticker, SecurityType.Equity, Market.USA);
            item.Time = item.Date;
            item.Value = (decimal) item.CustomAttribute1;
        }

        return new BaseDataCollection(endTime, config.Symbol, items);
    }
}
```

Initialize Custom Universes

To add a custom universe to your algorithm, in the `Initialize` method, pass your universe type and a selector function to the `AddUniverse` method. The selector function receives a list of your custom objects and must return a list of `Symbol` objects. In the selector function definition, you can use any of the properties of your custom data type. The `Symbol` objects that you return from the selector function set the constituents of the universe.

```C#
private Universe _universe;
// In Initialize
_universe = AddUniverse<MyCustomUniverseDataClass>("myCustomUniverse", Resolution.Daily, data => {
    return (from singleStockData in data
           where singleStockData.CustomAttribute1 > 0
           orderby singleStockData.CustomAttribute2 descending
           select singleStockData.Symbol).Take(5);
});
```

Historical Data

To get custom universe historical data, call the `History` method with the `Universe` object and the lookback period. The return type is a `IEnumerable<BaseDataCollection>` and you have to cast its items to `MyCustomUniverseDataClass` .

```
var history = History(_universe, 30);
foreach (var data in history)
{
    foreach (MyCustomUniverseDataClass singleStockData in data)
    {
        Log($"{singleStockData.Symbol} CustomAttribute1 at {singleStockData.EndTime}:
{singleStockData.CustomAttribute1}");
    }
}
```

Selection Frequency

Custom universes run on a schedule based on the `EndTime` of your custom data objects. To adjust the selection schedule, see Schedule .

Examples

The following examples demonstrate some common practices for Custom Universe.

Example 1: Sourcing from the Object Store

This project demonstrates how to read custom data from the Object Store , and then use it to define a universe and place trades. The following research environment file generates the demo universe data, which contains a daily set of assets and their respective signals:

```
// Load the assembly files and data types in their own cell.
#load "../Initialize.csx"
```

```
// Import the data types.
#load "../QuantConnect.csx"
using QuantConnect;
using QuantConnect.Research;
using QuantConnect.Data.Market;
using MathNet.Numerics.Distributions;

// Create a QuantBook.
var qb = new QuantBook();
// Make a history request to get a list of trading days.
var history = qb.History<TradeBar>(
    Symbol.Create("SPY", SecurityType.Equity, Market.USA),
    new DateTime(2015, 1, 1), new DateTime(2024, 12, 31), Resolution.Daily
).ToList();
// Create a variable to store the file contents.
var contents = new List<string>();
// Define the list of possible assets.
var tickers = new List<string> { "SPY", "TLT", "GLD", "USO", "IWM" };
// Loop through the list of trading days.
for (int i = 0; i < history.Count; i++)
{
    // Create a random number generator.
    var random = new Random(i);
    // Select 3 of the assets and define their target portfolio weights.
    var date = history[i].EndTime;
    var equities = tickers.OrderBy(x => random.Next()).Take(3).ToList();
    var weights = Dirichlet.Sample(random, new[] { 10d, 5d, 3d });
    // Append the target portfolio weights for this trading day to the file.
    contents.AddRange(new [] {
        $"{date:yyyy-MM-dd},{equities[0]},{weights[0]}",
        $"{date:yyyy-MM-dd},{equities[1]},{weights[1]}",
        $"{date:yyyy-MM-dd},{equities[2]},{weights[2]}"
    });
}
// Save the file to the Object Store.
Console.WriteLine(qb.ObjectStore.Save("portfolioTargets.csv", string.Join("\n", contents)));
```

The following algorithm file reads the preceding CSV file from the Object Store and uses its contents to form the daily universe and place trades:

```C#
public class CustomUniverseExampleAlgorithm : QCAlgorithm
{
    // Create a dictionary to hold the target weight of each asset.
    private Dictionary<Symbol, decimal> _weightBySymbol = new();
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add a universe that reads from the Object Store.
        _universe = AddUniverse<CustomUniverseData>("CustomUniverse", Resolution.Daily, (altCoarse) => {
            // Select the symbols that have a significant weight in the custom universe data to avoid
            // small-size trades that erode returns. Save the weight to use during the rebalance.
            _weightBySymbol = altCoarse
                .Select(d => d as CustomUniverseData)
                .Where(d => d.Weight > 0.05m)
                .ToDictionary(d => d.Symbol, d => d.Weight);
            return _weightBySymbol.Keys;
        });
        // Add a Scheduled Event to rebalanc\e the portfolio.
        var spy = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
        Schedule.On(
            DateRules.EveryDay(spy),
            TimeRules.AfterMarketOpen(spy, 1),
            () => SetHoldings(
                _universe.Selected.Select(symbol => new PortfolioTarget(symbol, _weightBySymbol[symbol])).ToList(),
                liquidateExistingHoldings: true
            )
        );
    }
}

public class CustomUniverseData : BaseData
{
    public decimal Weight;
    public override DateTime EndTime
    {
        // Set the data period to 1 day.
        get { return Time + QuantConnect.Time.OneDay; }
        set { Time = value - QuantConnect.Time.OneDay; }
    }

    public override SubscriptionDataSource GetSource(SubscriptionDataConfig config, DateTime date, bool isLiveMode)
    {
        // Define the location and format of the data file.
        return new SubscriptionDataSource(
            "portfolioTargets.csv", SubscriptionTransportMedium.ObjectStore, FileFormat.Csv
        );
    }

    public override BaseData Reader(SubscriptionDataConfig config, string line, DateTime date, bool isLiveMode)
    {
        // Skip the header row.
        if (!Char.IsDigit(line[0]))
        {
            return null;
        }
        // Split the line by each comma.
        var items = line.Split(",");

        // Parse the data from the CSV file.
        return new CustomUniverseData
        {
            EndTime = Parse.DateTimeExact(items[0], "yyyy-MM-dd"),
            Symbol = Symbol.Create(items[1], SecurityType.Equity, Market.USA),
            Weight = decimal.Parse(items[2], NumberStyles.Any, CultureInfo.InvariantCulture)
        };
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
DropboxUniverseSelectionAlgorithm.cs C# DropboxBaseDataUniverseSelectionAlgorithm.cs C#

Universes

Dataless Scheduled Universes

Introduction

A dataless scheduled universe let's you select a set of assets on a specific schedule. You can control which days the other types of universe run by adjusting the Schedule universe setting . However, the `Schedule` universe setting doesn't accept a `TimeRule` argument, so you can't control the time of the day they run. In contrast, a dataless scheduled universe accepts a `TimeRule` argument, but its selection function only receives the algorithm time.

Create Universes

To add a dataless scheduled universe, in the `Initialize` method, call the `AddUniverse` method with a `ScheduledUniverse` object.

```C#
public class MyUniverseAlgorithm : QCAlgorithm {
    private Universe _universe;
    public override void Initialize()
    {
        UniverseSettings.Asynchronous = true;
        _universe = AddUniverse(new ScheduledUniverse(DateRules.MonthStart(), TimeRules.At(8, 0), SelectSymbols));
    }


    private IEnumerable<Symbol> SelectSymbols(DateTime dt)
    {
        return new[] { QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA) };
    }
}
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`dateRule`	`IDateRule`	Date rule that defines what days the universe selection function runs	
`timeRule`	`ITimeRule`	Time rule that defines what times on each day selected by date rule the universe selection function runs	
`Func<DateTime, IEnumerable< Symbol>>`	`selector`	Selector function that accepts the current date time and returns the universe's `Symbol` objects	
`settings`	`UniverseSettings`	The universe settings . If you don't provide an argument, the model uses the `algorithm.UniverseSettings` by default.	`null`

Date Rules

The following table describes the supported `DateRules` :

Member	Description
`DateRules.SetDefaultTimeZone(DateTimeZone timeZone);`	Sets the time zone for the `DateRules` object used in all methods in this table. The default time zone is the algorithm time zone .
`DateRules.On(int year, int month, int day)`	Trigger an event on a specific date.
`DateRules.On(params DateTime[] dates)`	Trigger an event on specific dates.
`DateRules.EveryDay()`	Trigger an event all seven days of the week.
`DateRules.EveryDay(Symbol symbol, bool extendedMarketHours = false)`	Trigger an event every day a specific symbol is trading.
`DateRules.Every(params DayOfWeek[] days)`	Trigger an event on specific days throughout the week. To view the `DayOfWeek` enum members, see DayOfWeek Enum in the .NET documentation.
`DateRules.MonthStart(int daysOffset = 0)`	Trigger an event on the first day of each month plus an offset.
`DateRules.MonthStart(Symbol symbol, int daysOffset = 0)`	Trigger an event on the first tradable date of each month for a specific symbol plus an offset.
`DateRules.MonthEnd(int daysOffset = 0)`	Trigger an event on the last day of each month minus an offset.
`DateRules.MonthEnd(Symbol symbol, int daysOffset = 0)`	Trigger an event on the last tradable date of each month for a specific symbol minus an offset.
`DateRules.WeekStart(int daysOffset = 0)`	Trigger an event on the first day of each week plus an offset.
`DateRules.WeekStart(Symbol symbol, int daysOffset = 0)`	Trigger an event on the first tradable date of each week for a specific symbol plus an offset.
`DateRules.WeekEnd(int daysOffset = 0)`	Trigger an event on the last day of each week minus an offset.
`DateRules.WeekEnd(Symbol symbol, int daysOffset = 0)`	Trigger an event on the last tradable date of each week for a specific symbol minus an offset.
`DateRules.YearStart(int daysOffset = 0)`	Trigger an event on the first day of each year plus an offset.
`DateRules.YearStart(Symbol symbol, int daysOffset = 0)`	Trigger an event on the first tradable date of each year for a specific symbol plus an offset.
`DateRules.YearEnd(int daysOffset = 0)`	Trigger an event on the last day of each year minus an offset.
`DateRules.YearEnd(Symbol symbol, int daysOffset = 0)`	Trigger an event on the last tradable date of each year for a specific symbol minus an offset.
`DateRules.Today`	Trigger an event once today.
`DateRules.Tomorrow`	Trigger an event once tomorrow.

To define custom date rules, create a `FuncDateRule` object. The `FuncDateRule` constructor expects a `name` argument of type `string` and a `getDatesFunction` argument of type `Func<DateTime, DateTime, IEnumerable<DateTime>>` . The `getDatesFunction` function receives the start and end dates of the algorithm and returns a list of dates for the date rule. In live trading, the end date is 12/31/2025. The following example demonstrates how to define a date rule that represents the 10th day of each month:

```C#
// Create a date rule that specifies the 10th day of each month.
var dateRule = new FuncDateRule(
    name: "10th_day_of_the_month",
    getDatesFunction: (start, end) => Enumerable.Range(start.Year, end.Year - start.Year + 1)
        .SelectMany(year => Enumerable.Range(1, 12).Select(month => new DateTime(year, month, 10)))
);
```

Some date rules require a Symbol. However, they accept a string representation of the Symbol as well. If your algorithm doesn't add the security , it assumes the string refers to a US Equity Symbol. For example, if you use the `DateRules.EveryDay("ES")` method, it assumes that 'ES' refers to a US Equity. To create a Symbol without adding a security to your algorithm, call the `QuantConnect.Symbol.Create` method.

```C#
var symbol = QuantConnect.Symbol.Create("ES", SecurityType.Future, Market.CME);
```

Time Rules

The following table describes the supported `TimeRules` :

Member	Description
`TimeRules.SetDefaultTimeZone(DateTimeZone timeZone)`	Sets the time zone for the `TimeRules` object used in all methods in this table, except when a different time zone is given. The default time zone is the algorithm time zone .
`TimeRules.BeforeMarketOpen(Symbol symbol, double minutesBeforeOpen = 0, bool extendedMarketOpen = false)`	Trigger an event a few minutes before market open for a specific symbol (default is 0). This rule doesn't work for Crypto securities or custom data.
`TimeRules.AfterMarketOpen(Symbol symbol, double minutesAfterOpen = 0, bool extendedMarketOpen = false)`	Trigger an event a few minutes after market open for a specific symbol (default is 0). This rule doesn't work for Crypto securities or custom data.
`TimeRules.BeforeMarketClose(Symbol symbol, double minutesBeforeClose = 0, bool extendedMarketOpen = false)`	Trigger an event a few minutes before market close for a specific symbol (default is 0). This rule doesn't work for Crypto securities or custom data.
`TimeRules.AfterMarketClose(Symbol symbol, double minutesAfterClose = 0, bool extendedMarketOpen = false)`	Trigger an event a few minutes after market close for a specific symbol (default is 0). This rule doesn't work for Crypto securities or custom data.
`TimeRules.Every(TimeSpan interval)`	Trigger an event every period interval starting at midnight. This time rule triggers every period, regardless of whether or not the market is open.
`TimeRules.Now`	Trigger an event at the current time of day.
`TimeRules.Midnight`	Trigger an event at midnight.
`TimeRules.Noon`	Trigger an event at noon.
`TimeRules.At(int hour, int minute, int second = 0)`	Trigger an event at a specific time of day (e.g. 13:10).
`TimeRules.At(int hour, int minute, int second, DateTimeZone timeZone)`	Trigger an event at a specific time of day in the given time zone (e.g. 13:10 UTC).

To define custom time rules, create a `FuncTimeRule` object. The `FuncTimeRule` constructor expects a `name` argument of type `string` and a `createUtcEventTimesFunction` argument of type `Func<IEnumerable<DateTime>, IEnumerable<DateTime>>` . The function receives the list of dates from the date rule and then returns a list of `DateTime` that define the time rule.

```
C#
var timeRule = new FuncTimeRule(
    name: "CustomTimeRule",
    createUtcEventTimesFunction: dates => dates.Select(d => d.AddHours(10)));
```

Some time rules require a Symbol. However, they accept a string representation of the Symbol as well. If your algorithm doesn't add the security , it assumes the string refers to a US Equity Symbol. For example, if you use the `TimeRules.BeforeMarketOpen("ES", 15)` method, it assumes that 'ES' refers to a US Equity. To create a Symbol without adding a security to your algorithm, call the `QuantConnect.Symbol.Create` method.

```
C#
var symbol = QuantConnect.Symbol.Create("ES", SecurityType.Future, Market.CME);
```

Selection Frequency

Dataless scheduled universes run at whatever selection schedule you define by the `dateRule` and `timeRule` arguments.

Examples

The following examples demonstrate some common practices for dataless scheduled universes.

Example 1: Download External Universe Files

The following algorithm downloads a CSV file from Dropbox. The first column in the file contains the universe date. The second column in the file contains a list of Equity tickers that represents the universe for the day. The dataless scheduled universe parses the file contents each trading day and adds the universe constituents.

C#

```
public class DatalessScheduledUniverseExampleAlgorithm : QCAlgorithm
{
    // Create a dictionary for the universe data where the key is the date and
    // the value is a comma-separated string of stock tickers.
    private Dictionary<DateTime, string> _tickersByDate = new();
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Download the initial universe file.
        DownloadUniverseFile();
        // Add the custom universe.
        UniverseSettings.ExtendedMarketHours = true;
        var spy = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
        var dateRule = DateRules.EveryDay(spy);
        _universe = AddUniverse(new ScheduledUniverse(dateRule, TimeRules.At(8, 0), SelectAssets));
        // Schedule rebalances for market open.
        Schedule.On(dateRule, TimeRules.AfterMarketOpen(spy, 0), Rebalance);
    }

    private void DownloadUniverseFile()
    {
        // Download the universe CSV file. Dropbox links require the "dl=1" URL parameter.
        var file = Download(
            "https://www.dropbox.com/scl/fi/fbrxitk4ec3w91nse1raa/df.csv?
rlkey=7r042rukzkthp7y1srloyhkov&st=5r4sdfwd&dl=1"
        );
        // Convert the CSV file data into a dictionary where the key is the date and
        // the value is a comma-separated string of stock tickers.
        foreach (var line in file.Split('\n').Skip(1))
        {
            // Skip empty lines.
            if (line.IsNullOrEmpty())
            {
                continue;
            }
            var items = line.Split("\"");
            var date = Parse.DateTimeExact(items[0].Split(",")[0], "yyyy-MM-dd").Date;
            _tickersByDate[date] = items[1];
        }
    }

    private IEnumerable<Symbol> SelectAssets(DateTime dt)
    {
        // When live trading, re-download the CSV file each day to get the new rows.
        if (LiveMode)
        {
            DownloadUniverseFile();
        }
        // Get the current day's data from the CSV file.
        if (!_tickersByDate.TryGetValue(dt.Date, out var tickers))
        {
            // If there isn't an entry for the current date, return an empty universe.
            return Enumerable.Empty<Symbol>();
        }
        // Convert the stock tickers in the CSV file to Symbol objects.
        return tickers
            .Split(',')
            .Select(x => QuantConnect.Symbol.Create(x, SecurityType.Equity, Market.USA));
    }

    private void Rebalance()
    {
        // Form an equal-weighted portfolio of all the universe constituents.
        var weight = 1m / _universe.Selected.Count;
        SetHoldings(_universe.Selected.Select(symbol => new PortfolioTarget(symbol, weight)).ToList(),
liquidateExistingHoldings: true);
    }
}
```

Example 2: Quarter End Selection

The following algorithm selects SPY on the last month of each quarter. For the remaining months, it selects no assets.

```C#
public class DatalessScheduledUniverseDemoAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        AddUniverse(new ScheduledUniverse(
            DateRules.MonthStart(),
            TimeRules.At(8, 0),
            (dt) =>
            {
                if (dt.Month % 3 == 0)
                {
                    return new[] { QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA) };
                }
                return Enumerable.Empty<Symbol>();
            }
        ));
    }
}
```

Example 3: Third Week VIX

Standard Options expire at end of the third week of each month. The following algorithm selects VIX-related products on the third week to trade the foreseeable increase in volatility.

```C#
public class DatalessScheduledUniverseDemoAlgorithm : QCAlgorithm
{
    private int _month = -1;
    private int _week = -1;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        AddUniverse(new ScheduledUniverse(
            DateRules.WeekStart(),
            TimeRules.At(8, 0),
            (dt) =>
            {
                if (dt.Month == _month)
                {
                    if (++_week == 3)
                    {
                        return new[] { QuantConnect.Symbol.Create("VXZ", SecurityType.Equity, Market.USA) };
                    }
                    return Enumerable.Empty<Symbol>();
                }
                _month = dt.Month;
                _week = 0;
                return Enumerable.Empty<Symbol>();
            }
        ));
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
CustomDataUniverseScheduledRegressionAlgorithm.cs C#

Datasets

Datasets

Overview

Introduction

It's common to use price and fundamental data points to inform trading decisions. Alternative datasets are datasets that aren't classified as market or fundamental data. Alternative data research began gaining popularity in 2007 after the financial crisis and its usage has continued to grow over the years. As usage has grown, so has the number of unique alternative datasets that are available for investors. The benefit of alternative datasets is that it’s easier to find alpha in these datasets because they receive less attention from investment researchers and these datasets contain information not found in market and fundamental datasets. Most traders research with market and fundamental datasets, so it's difficult to find alpha in them.

Methodology

The journey of raw data from its source to its integration with a trading platform can be complex. Alternative data is generally created through individuals, business processes, and sensors. Individuals create content on social media platforms, write reviews on e-commerce sites, and visit corporate Wikipedia pages. Businesses process transactions, file SEC reports, and issue company announcements. Sensors collect location data, gather satellite images, and monitor customer behavior.

Data providers can capture these forms of data for traders to inform trading decisions in ways that supplement traditional market and fundamental datasets. With over a million terabytes of data on the internet, there is virtually a limitless amount of information that can be informative for investors when analyzing a security’s performance in the marketplace.

Examples

A common example of alternative data in a trading system is using sentiment analysis on news sources to predict the future price movements of Equities. A second example is using announcements regarding corporate share buyback programs to trading Equities. A third example is using global supply and demand information for US Crude Products to inform Futures trades. To view all of the datasets we have available, see the left navigation menu.

Challenges

A challenge of working with datasets is that it can be difficult to find a reliable, high-quality, and inexpensive data source to use in live algorithms. If a data source has survivorship bias, insufficient history, or hasn’t been cleaned and processed properly, the data can cause poor trading performance. We thoroughly vet the datasets we add to the Dataset Market to ensure they are free of survivorship bias and have a reliable live feed. All of the available datasets have already been cleaned and processed, so you can just focus on the strategy research and development.

Future Outlook

The Dataset Market is still in the early phases of development. Over time, we will add more markets and alternative datasets. Alternative datasets are an ever-expanding data source, so it’s expected to continue to grow into the future. In today’s digital world, more data is created each year than in the previous year. To take advantage of the trend, we have implemented a new

onboarding process to integrate many new alternative data sources into the QuantConnect platform. With the Dataset Market, you can import a new alternative dataset into your trading algorithm with just a single line of code. The Dataset Market gives everyone the ability to access high-quality alternative datasets that were previously only accessible to major hedge funds.

Live Trading Considerations

We receive most daily alternative data at 7 AM Eastern Time (ET) and start processing it. In live trading, your algorithm receives this data within 30 minutes of 7 AM ET. In backtesting, this data has a timestamp of midnight, so your algorithm receives it about seven hours earlier than it does in live mode.

In backtests, your algorithm receives data at perfect timing. If you request minute resolution data, your algorithm receives the bars at the top of each minute. In live trading, bars have a slight delay, so you may receive them milliseconds after the top of each minute.

Note that there will be delay for data available to be backtested. Live feed data is available for backtesting after 24-48 hours normally. Consider using Paper live trading if close monitoring is needed.

Datasets

QuantConnect

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Binance Crypto Future Margin Rate Data

Bybit Crypto Future Margin Rate Data

CFD Data

Cash Indices

FOREX Data

Fear and Greed

International Future Universe

US ETF Constituents

US Equities Short Availability

US Equity Coarse Universe

US Equity Option Universe

US Equity Security Master

US Future Option Universe

US Future Universe

US Futures Security Master

US Index Option Universe

QuantConnect

Binance Crypto Future Margin Rate Data

Introduction

The Binance Crypto Future Margin Rate Data by QuantConnect is for crypto-currency futures margin interest data points. The data covers 718 Cryptocurrency pairs, starts in August 2020, and is delivered on a daily update frequency. This dataset is created by downloading data using Binance API.

This dataset is an important companion to the Binance Crypto Future Price Data dataset because it contains information on margin interest data to model margin costs.

For more information about the Binance Crypto Future Margin Rate Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Getting Started

The following snippet demonstrates how to request data from the Binance Crypto Future Margin Rate dataset:

```C#
private Symbol _cryptoFutureSymbol;

public override void Initialize()
{
    SetBrokerageModel(BrokerageName.BinanceFutures, AccountType.Margin);
    //SetBrokerageModel(BrokerageName.BinanceCoinFutures, AccountType.Margin);

    // perpetual futures does not have a filter function
    _cryptoFutureSymbol = AddCryptoFuture("BTCBUSD", Resolution.Minute).Symbol;
}
```

The Binance Crypto Future Margin Rate data is added to your algorithm along with the market data when you add a crypto future subscription.

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	August 2020
Asset Coverage	718 Crypto Futures Pairs
Data Density	Regular
Resolution	Daily
Timezone	UTC
Market Hours	Always Open

Requesting Data

To add Binance Crypto Future Margin Rate data to your algorithm, call the `AddCryptoFuture` method. Save a reference to the Crypto Future `Symbol` so you can access the data later in your algorithm.

```
C#
public class CoinAPIDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol ;

    public override void Initialize()
    {
        SetStartDate(2020, 6, 1);
        SetEndDate(2021, 6, 1);

        // Set Account Currency to Binance Stable Coin for USD
        SetAccountCurrency("BUSD");
        SetCash(100000);

        SetBrokerageModel(BrokerageName.BinanceFutures, AccountType.Margin);
        SetBrokerageModel(BrokerageName.BinanceCoinFutures, AccountType.Margin);

        var cryptoFuture = AddCryptoFuture("BTCBUSD", Resolution.Minute);
        // perpetual futures does not have a filter function
        _symbol = cryptoFuture.Symbol;
    }
}
```

For more information about creating Crypto Future subscriptions, see Requesting Data .

Accessing Data

To get the current Binance Crypto Margin Rate data, index the `MarginInterestRates` property of the current `Slice` with the Crypto Future `Symbol` . `Slice` objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.MarginInterestRates.ContainsKey(_symbol))
    {
        var interestRate = slice.MarginInterestRates[_symbol].InterestRate;
        Log($"{_symbol} price at {slice.Time}: {interestRate}");
    }
}
```

You can also iterate through all of the data objects in the current `Slice` .

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.MarginInterestRates)
    {
        var symbol = kvp.Key;
        var marginInterestRate = kvp.Value;
        var interestRate = marginInterestRate.InterestRate;
        Log($"{symbol} price at {slice.Time}: {interestRate}");
    }
}
```

For more information about accessing Crypto Future data, see Handling Data .

Historical Data

To get historical Binance Crypto Future Margin Rate data, call the `History<MarginInterestRate>` method with the Crypto Future `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
var history = History<MarginInterestRate>(_symbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To unsubscribe from a Crypto pair that you added with the `AddCrypto` method, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_symbol);
```

The `RemoveSecurity` method cancels your open orders for the security and liquidates your holdings in the virtual pair .

Supported Assets

The following table shows the available Crypto Future pairs:

Pairs Available (718)					
0GUSDT	1000000BOBUSDT	1000000MOGUSDT	1000BONKUSDC	1000BONKUSDT	1000CATUSDT
1000CHEEMSUSDT	1000FLOKIUSDT	1000LUNCBUSD	1000LUNCUSDT	1000PEPEUSDC	1000PEPEUSDT
1000RATSUSDT	1000SATSUSDT	1000SHIBBUSD	1000SHIBUSDC	1000SHIBUSDT	1000WHYUSDT
1000XECUSDT	1000XUSDT	1INCHUSDT	1MBABYDOGEUSDT	2ZUSDT	4USDT
A2ZUSDT	AAVEUSD	AAVEUSDC	AAVEUSDT	ACEUSDT	ACHUSDT
ACTUSDT	ACXUSDT	ADABUSD	ADAUSD	ADAUSDC	ADAUSDT
AERGOUSDT	AEROUSDT	AEVOUSDT	AGIXBUSD	AGIXUSDT	AGLDUSDT
AGTUSDT	AI16ZUSDT	AIAUSDT	AINUSDT	AIOTUSDT	AIOUSDT
AIUSDT	AIXBTUSDT	AKEUSDT	AKTUSDT	ALCHUSDT	ALGOUSD

ALGOUSDT	ALICEUSDT	ALLUSDT	ALPACAUSDT	ALPHAUSDT	ALPINEUSDT
ALTUSDT	AMBBUSD	AMBUSDT	ANCBUSD	ANIMEUSDT	ANKRUSDT
ANTUSDT	APEBUSD	APEUSD	APEUSDT	API3USDT	APTBUSD
APTUSD	APTUSDT	ARBUSDC	ARBUSDT	ARCUSDT	ARIAUSDT
ARKMUSDT	ARKUSDT	ARPAUSDT	ARUSDT	ASRUSDT	ASTERUSDT
ASTRUSDT	ATAUSDT	ATHUSDT	ATOMUSD	ATOMUSDT	AUCTIONBUSD
AUCTIONUSDT	AUDIOUSDT	AUSDT	AVAAIUSDT	AVAUSDT	AVAXBUSD
AVAXUSD	AVAXUSDC	AVAXUSDT	AVNTUSDT	AWEUSDT	AXLUSDT
AXSUSD	AXSUSDT	B2USDT	B3USDT	BABYUSDT	BADGERUSDT
BAKEUSDT	BALUSDT	BANANAS31USDT	BANANAUSDT	BANDUSDT	BANKUSDT
BANUSDT	BARDUSDT	BASUSDT	BATUSDT	BBUSDT	BCHUSD
BCHUSDC	BCHUSDT	BDXNUSDT	BEAMXUSDT	BELUSDT	BERAUSDT
BICOUSDT	BIDUSDT	BIGTIMEUSDT	BIOUSDC	BIOUSDT	BLESSUSDT
BLUAIUSDT	BLUEBIRDUSDT	BLURUSDT	BLZUSDT	BMTUSDT	BNBBUSD
BNBUSD	BNBUSDC	BNBUSDT	BNTUSDT	BNXUSDT	BOMEUSDC
BOMEUSDT	BONDUSDT	BRETTUSDT	BROCCOLI714USDT	BROCCOLIF3BUSDT	BRUSDT
BSVUSDT	BSWUSDT	BTCBUSD	BTCDOMUSDT	BTCSTUSDT	BTCUSD
BTCUSDC	BTCUSDT	BTRUSDT	BTSUSDT	BULLAUSDT	BUSDT
C98USDT	CAKEUSDT	CARVUSDT	CATIUSDT	CELOUSDT	CELRUSDT
CETUSUSDT	CFXUSDT	CGPTUSDT	CHESSUSDT	CHILLGUYUSDT	CHRUSDT
CHZUSD	CHZUSDT	CKBUSDT	CLOUSDT	COAIUSDT	COCOSUSDT
COMBOUSDT	COMPUSDT	COOKIEUSDT	COSUSDT	COTIUSDT	COWUSDT
CROSSUSDT	CRVUSDC	CRVUSDT	CTKUSDT	CTSIUSDT	CUDISUSDT
CUSDT	CVCUSDT	CVXBUSD	CVXUSDT	CYBERUSDT	DAMUSDT
DARUSDT	DASHUSDT	DEEPUSDT	DEFIUSDT	DEGENUSDT	DEGOUSDT
DENTUSDT	DEXEUSDT	DFUSDT	DGBUSDT	DIAUSDT	DMCUSDT
DODOBUSD	DODOXUSDT	DOGEBUSD	DOGEUSD	DOGEUSDC	DOGEUSDT
DOGSUSD	DOGSUSDT	DOLOUSDT	DOODUSDT	DOTBUSD	DOTUSD

DOTUSDT	DRIFTUSDT	DUSDT	DUSKUSDT	DYDXUSDT	DYMUSDT
EDENUSDT	EDUUSDT	EGLDUSD	EGLDUSDT	EIGENUSDT	ENAUSDC
ENAUSDT	ENJUSDT	ENSOUSDT	ENSUSD	ENSUSDT	EOSUSD
EOSUSDT	EPICUSDT	EPTUSDT	ERAUSDT	ESPORTSUSDT	ETCBUSD
ETCUSD	ETCUSDT	ETHBTC	ETHBUSD	ETHFIUSDC	ETHFIUSDT
ETHUSD	ETHUSDC	ETHUSDT	ETHWUSDT	EULUSDT	EVAAUSDT
FARTCOINUSDT	FETUSDT	FFUSDT	FHEUSDT	FIDAUSDT	FILBUSD
FILUSD	FILUSDC	FILUSDT	FIOUSDT	FISUSDT	FLMUSDT
FLOCKUSDT	FLOWUSDT	FLUIDUSDT	FLUXUSDT	FOOTBALLUSDT	FORMUSDT
FORTHUSDT	FRONTUSDT	FTMBUSD	FTMUSD	FTMUSDT	FTTBUSD
FTTUSDT	FUNUSDT	FUSDT	FXSUSDT	GALABUSD	GALAUSD
GALAUSDT	GALBUSD	GALUSDT	GASUSDT	GHSTUSDT	GIGGLEUSDT
GLMRUSDT	GLMUSDT	GMTBUSD	GMTUSD	GMTUSDT	GMXUSDT
GOATUSDT	GPSUSDT	GRASSUSDT	GRIFFAINUSDT	GRTUSDT	GTCUSDT
GUNUSDT	GUSDT	HAEDALUSDT	HANAUSDT	HBARUSDC	HBARUSDT
HEIUSDT	HEMIUSDT	HFTUSDT	HIFIUSDT	HIGHUSDT	HIPPOUSDT
HIVEUSDT	HMSTRUSDT	HNTUSDT	HOLOUSDT	HOMEUSDT	HOOKUSDT
HOTUSDT	HUMAUSDT	HUSDT	HYPERUSDT	HYPEUSDT	ICNTUSDT
ICPBUSD	ICPUSDT	ICXUSD	ICXUSDT	IDEXUSDT	IDOLUSDT
IDUSDT	ILVUSDT	IMXUSDT	INITUSDT	INJUSDT	INUSDT
IOSTUSDT	IOTAUSDT	IOTXUSDT	IOUSDT	IPUSDC	IPUSDT
JASMYUSDT	JELLYJELLYUSDT	JOEUSDT	JSTUSDT	JTOUSDT	JUPUSDT
KAIAUSDT	KAITOUSDC	KAITOUSDT	KASUSDT	KAVAUSDT	KDAUSDT
KERNELUSDT	KEYUSDT	KGENUSDT	KLAYUSDT	KMNOUSDT	KNCUSD
KNCUSDT	KOMAUSDT	KSMUSDT	LABUSDT	LAUSDT	LAYERUSDT
LDOBUSD	LDOUSDT	LEVERBUSD	LEVERUSDT	LIGHTUSDT	LINAUSDT
LINEAUSDT	LINKBUSD	LINKUSD	LINKUSDC	LINKUSDT	LISTAUSDT
LITUSDT	LOKAUSDT	LOOMUSDT	LPTUSDT	LQTYUSDT	LRCUSDT

LSKUSDT	LTCBUSD	LTCUSD	LTCUSDC	LTCUSDT	LUMIAUSDT
LUNA2BUSD	LUNA2USDT	LYNUSDT	MAGICUSDT	MANAUSD	MANAUSDT
MANTAUSDT	MASKUSDT	MATICBUSD	MATICUSD	MATICUSDC	MATICUSDT
MAVIAUSDT	MAVUSDT	MBLUSDT	MBOXUSDT	MDTUSDT	MELANIAUSDT
MEMEFIUSDT	MEMEUSDT	MERLUSDT	METISUSDT	METUSDT	MEUSDT
MEWUSDT	MILKUSDT	MINAUSDT	MIRAUSDT	MITOUSDT	MKRUSDT
MLNUSDT	MOCAUSDT	MONUSDT	MOODENGUSDT	MORPHOUSDT	MOVEUSDT
MOVRUSDT	MTLUSDT	MUBARAKUSDT	MUSDT	MYROUSDT	MYXUSDT
NAORISUSDT	NEARBUSD	NEARUSD	NEARUSDC	NEARUSDT	NEIROETHUSDT
NEIROUSDT	NEOUSDC	NEOUSDT	NEWTUSDT	NFPUSDT	NILUSDT
NKNUSDT	NMRUSDT	NOMUSDT	NOTUSDT	NTRNUSDT	NULSUSDT
NXPCUSDT	OBOLUSDT	OCEANUSDT	OGNUSDT	OGUSDT	OLUSDT
OMGUSDT	OMNIUSDT	OMUSDT	ONDOUSDT	ONEUSDT	ONGUSDT
ONTUSDT	OPENUSDT	OPUSD	OPUSDT	ORBSUSDT	ORCAUSDT
ORDERUSDT	ORDIUSDC	ORDIUSDT	OXTUSDT	PARTIUSDT	PAXGUSDT
PENDLEUSDT	PENGUUSDC	PENGUUSDT	PEOPLEUSDT	PERPUSDT	PHAUSDT
PHBBUSD	PHBUSDT	PIPPINUSDT	PIXELUSDT	PLAYUSDT	PLUMEUSDT
PNUTUSDC	PNUTUSDT	POLUSDT	POLYXUSDT	PONKEUSDT	POPCATUSDT
PORT3USDT	PORTALUSDT	POWRUSDT	PROMPTUSDT	PROMUSDT	PROVEUSDT
PTBUSDT	PUFFERUSDT	PUMPBTCUSDT	PUMPUSDT	PUNDIXUSDT	PYTHUSDT
QNTUSDT	QTUMUSDT	QUICKUSDT	QUSDT	RADUSDT	RAREUSDT
RAYSOLUSDT	RAYUSDT	RDNTUSDT	RECALLUSDT	REDUSDT	REEFUSDT
REIUSDT	RENDERUSDT	RENUSDT	RESOLVUSDT	REZUSDT	RIFUSDT
RIVERUSDT	RLCUSDT	RNDRUSDT	RONINUSDT	ROSEUSD	ROSEUSDT
RPLUSDT	RSRUSDT	RUNEUSD	RUNEUSDT	RVNUSDT	RVVUSDT
SAFEUSDT	SAGAUSDT	SAHARAUSDT	SANDBUSD	SANDUSD	SANDUSDT
SANTOSUSDT	SAPIENUSDT	SCRTUSDT	SCRUSDT	SCUSDT	SEIUSDT
SFPUSDT	SHELLUSDT	SIGNUSDT	SIRENUSDT	SKATEUSDT	SKLUSDT

SKYAIUSDT	SKYUSDT	SLERFUSDT	SLPUSDT	SNTUSDT	SNXUSDT
SOLBUSD	SOLUSD	SOLUSDC	SOLUSDT	SOLVUSDT	SOMIUSDT
SONICUSDT	SOONUSDT	SOPHUSDT	SPELLUSDT	SPKUSDT	SPXUSDT
SQDUSDT	SRMUSDT	SSVUSDT	STBLUSDT	STEEMUSDT	STGUSDT
STMXUSDT	STORJUSDT	STOUSDT	STPTUSDT	STRAXUSDT	STRKUSDT
STXUSDT	SUIUSD	SUIUSDC	SUIUSDT	SUNUSDT	SUPERUSDT
SUSDT	SUSHIUSDT	SWARMSUSDT	SWELLUSDT	SXPUSDT	SXTUSDT
SYNUSDT	SYRUPUSDT	SYSUSDT	TACUSDT	TAGUSDT	TAIKOUSDT
TAKEUSDT	TANSSIUSDT	TAOUSDT	TAUSDT	THETAUSD	THETAUSDT
THEUSDT	TIAUSDC	TIAUSDT	TLMBUSD	TLMUSDT	TNSRUSDT
TOKENUSDT	TOMOUSDT	TONUSDT	TOSHIUSDT	TOWNSUSDT	TRADOORUSDT
TRBUSDT	TREEUSDT	TROYUSDT	TRUMPUSDC	TRUMPUSDT	TRUTHUSDT
TRUUSDT	TRXBUSD	TRXUSD	TRXUSDT	TSTUSDT	TURBOUSDT
TUSDT	TUTUSDT	TWTUSDT	UBUSDT	UMAUSDT	UNFIUSDT
UNIBUSD	UNIUSD	UNIUSDC	UNIUSDT	USDCUSDT	USELESSUSDT
USTCUSDT	USUALUSDT	UXLINKUSDT	VANAUSDT	VANRYUSDT	VELODROMEUSDT
VELVETUSDT	VETUSD	VETUSDT	VFYUSDT	VICUSDT	VIDTUSDT
VINEUSDT	VIRTUALUSDT	VOXELUSDT	VTHOUSDT	VVVUSDT	WALUSDT
WAVESBUSD	WAVESUSDT	WAXPUSDT	WCTUSDT	WIFUSD	WIFUSDC
WIFUSDT	WLDUSD	WLDUSDC	WLDUSDT	WLFIUSDC	WLFIUSDT
WOOUSDT	WUSDT	XAIUSDT	XANUSDT	XCNUSDT	XEMUSDT
XLMUSD	XLMUSDT	XMRUSD	XMRUSDT	XNYUSDT	XPINUSDT
XPLUSDT	XRPBUSD	XRPUSD	XRPUSDC	XRPUSDT	XTZUSD
XTZUSDT	XVGUSDT	XVSUSDT	YALAUSDT	YBUSDT	YFIUSDT
YGGUSDT	ZBTUSDT	ZECUSDT	ZENUSDT	ZEREBROUSDT	ZETAUSDT
ZILUSD	ZILUSDT	ZKCUSDT	ZKJUSDT	ZKUSDT	ZORAUSDT
ZRCUSDT	ZROUSDT	ZRXUSDT	[illegible]USDT		

Example Applications

The Binance Crypto Future Margin Rate dataset enables correct margin cost so you can accurately design strategies for Cryptocurrencies with term structure. Examples include the following strategies:

- Horizontal/Diagonal arbitrage with the underlying cryptocurrencies
- Trade Contango/Backwardation predictions
- Hedge for illiquid cryptocurrencies

Classic Algorithm Example

The following example algorithm holds a long position in the BTCUSDT perpetual Future contract on the Binance exchange. When the margin interest rate is relatively low, it flips to a short position.

```C#
public class BinanceCryptoFutureDataAlgorithm : QCAlgorithm
{
    private CryptoFuture _future;
    // Add an indicator to track the low of trailing margin interest rates.
    private Minimum _lowInterestRate = new(10);

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Set Account Currency to USDT, since USD cannot be used to trade.
        SetCash("USDT", 100000);
        // Binance Futures Exchange accepts both Cash and Margin account types, select the one you need for the best
reality modeling.
        SetBrokerageModel(BrokerageName.BinanceFutures, AccountType.Margin);
        // Requesting data, we only trade on BTCUSDT Future in Binance Future exchange
        // Perpetual futures does not have a filter function.
        _future = AddCryptoFuture("BTCUSDT", Resolution.Daily);
        // Historical data
        var history = History<MarginInterestRate>(_future.Symbol, 10, Resolution.Daily);
        foreach (var dataPoint in history)
        {
            _lowInterestRate.Update(dataPoint.EndTime, dataPoint.InterestRate);
        }
    }

    public override void OnData(Slice slice)
    {
        // Note that you may want to access the margin interest of the crypto future to calculate if it would impact
a trade's PnL
        // Or you can calculate the trade size on keeping the quote currency constant
        if (!slice.MarginInterestRates.TryGetValue(_future.Symbol, out var dataPoint))
        {
            return;
        }
        var interestRate = dataPoint.InterestRate;
        Plot("Margin Interest Rate", _future.Symbol.ToString(), interestRate);
        // Short when margin interest rates are relatively low. Otherwise, long.
        if (interestRate <= _lowInterestRate)
        {
            if (!_future.Holdings.IsShort)
            {
                SetHoldings(_future.Symbol, -0.5);
            }
        }
        else if (!_future.Holdings.IsLong)
        {
            SetHoldings(_future.Symbol, 1);
        }
        // Update the indicator.
        _lowInterestRate.Update(Time, interestRate);
    }
}
```

Framework Algorithm Example

The following example algorithm holds a long position in the BTCUSDT perpetual Future contract on the Binance exchange.

When the margin interest rate is relatively low, it flips to a short position.

```C#
public class BinanceCryptoFutureDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        Settings.FreePortfolioValuePercentage = 0.1m;
        // Set Account Currency to USDT, since USD cannot be used to trade.
        SetCash("USDT", 100000);
        // Binance Futures Exchange accepts both Cash and Margin account types, select the one you need for the best
reality modeling.
        SetBrokerageModel(BrokerageName.BinanceFutures, AccountType.Margin);

        UniverseSettings.Resolution = Resolution.Daily;
        UniverseSettings.Leverage = 2;
        // We only trade on BTCUSDT Future in Binance Future exchange
        var symbols = new List<Symbol>{
            QuantConnect.Symbol.Create("BTCUSDT", SecurityType.CryptoFuture, Market.Binance)
        };
        AddUniverseSelection(new ManualUniverseSelectionModel(symbols));
        // Custom alpha model to emit insights based on the Crypto Future price data
        AddAlpha(new CryptoFutureAlphaModel());
        // Equally invest to evenly dissipate the capital concentration risk of inidividual crypto pair
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
        SetExecution(new ImmediateExecutionModel());
    }
}

public class CryptoFutureAlphaModel : AlphaModel
{
    private Dictionary<Symbol, Minimum> _minBySymbol = new();

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = new List<Insight>();
        foreach (var kvp in _minBySymbol)
        {
            var symbol = kvp.Key;
            var lowInterestRate = kvp.Value;
            // Note that you may want to access the margin interest of the crypto future to calculate if it would
impact a trade's PnL
            // Or you can calculate the trade size on keeping the quote currency constant
            if (!slice.MarginInterestRates.TryGetValue(symbol, out var dataPoint))
                continue;
            var interestRate = dataPoint.InterestRate;
            algorithm.Plot("Margin Interest Rate", symbol.ToString(), interestRate);
            // Short when margin interest rates are relatively low. Otherwise, long.
            var direction = interestRate <= lowInterestRate.Current.Value
                ? InsightDirection.Down
                : InsightDirection.Up;
            insights.Add(Insight.Price(symbol, TimeSpan.FromDays(1), direction));
            // Update the indicator.
            lowInterestRate.Update(slice.Time, interestRate);
        }
        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            var symbol = security.Symbol;
            // Add an indicator to track the low of trailing margin interest rates.
            var min = new Minimum(10);
            _minBySymbol[symbol] = min;
            // Historical data
            var history = algorithm.History<MarginInterestRate>(symbol, 10, Resolution.Daily);
            foreach (var dataPoint in history)
            {
                min.Update(dataPoint.EndTime, dataPoint.InterestRate);
            }
        }
        foreach (var security in changes.RemovedSecurities)
        {
            _minBySymbol.Remove(security.Symbol);
        }
    }
```

```
}
```

Data Point Attributes

The Binance Crypto Future Margin Rate dataset provides `MarginInterestRates` objects with the following attributes:

QuantConnect

Bybit Crypto Future Margin Rate Data

Introduction

The Bybit Crypto Future Margin Rate Data by QuantConnect is for Cryptocurrency Futures margin interest data points. The data covers 631 Cryptocurrency pairs, starts in August 2020, and is delivered on a daily update frequency. This dataset is created by downloading data using Bybit API.

This dataset is an important companion to the Bybit Crypto Future Price Data dataset because it contains information on margin interest data to model margin costs.

For more information about the Bybit Crypto Future Margin Rate Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Getting Started

The following snippet demonstrates how to request data from the Bybit Crypto Future Margin Rate dataset:

```C#
private Symbol _cryptoFutureSymbol;

public override void Initialize
{
    SetBrokerageModel(BrokerageName.Bybit, AccountType.Margin);

    // perpetual futures does not have a filter function
    _cryptoFutureSymbol = AddCryptoFuture("BTCBUSD", Resolution.Minute).Symbol;
}
```

The Bybit Crypto Future Margin Rate data is added to your algorithm along with the market data when you add a crypto future subscription.

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	August 2020
Asset Coverage	631 Crypto Futures Pairs
Data Density	Regular
Resolution	Daily
Timezone	UTC
Market Hours	Always Open

Requesting Data

To add Bybit Crypto Future Margin Rate data to your algorithm, call the `AddCryptoFuture` method. Save a reference to the Crypto Future `Symbol` so you can access the data later in your algorithm.

```
C#
public class CoinAPIDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol ;

    public override void Initialize()
    {
        SetStartDate(2020, 6, 1);
        SetEndDate(2021, 6, 1);

        // Set Account Currency to Tether
        SetAccountCurrency("USDT", 100000);

        SetBrokerageModel(BrokerageName.Bybit, AccountType.Margin);

        var cryptoFuture = AddCryptoFuture("BTCUSDT", Resolution.Minute);
        // perpetual futures does not have a filter function
        _symbol = cryptoFuture.Symbol;
    }
}
```

For more information about creating Crypto Future subscriptions, see Requesting Data .

Accessing Data

To get the current Bybit Crypto Margin Rate data, index the `MarginInterestRates` property of the current `Slice` with the Crypto Future `Symbol` . `Slice` objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.MarginInterestRates.ContainsKey(_symbol))
    {
        var interestRate = slice.MarginInterestRates[_symbol].InterestRate;
        Log($"{_symbol} price at {slice.Time}: {interestRate}");
    }
}
```

You can also iterate through all of the data objects in the current `Slice` .

```
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.MarginInterestRates)
    {
        var symbol = kvp.Key;
        var marginInterestRate = kvp.Value;
        var interestRate = marginInterestRate.InterestRate;
        Log($"{symbol} price at {slice.Time}: {interestRate}");
    }
}
```

For more information about accessing Crypto Future data, see Handling Data .

Historical Data

To get historical Bybit Crypto Future Margin Rate data, call the `History<MarginInterestRate>` method with the Crypto Future `Symbol` . If there is no data in the period you request, the history result is empty.

```
var history = History<MarginInterestRate>(_symbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To unsubscribe from a Crypto pair that you added with the `AddCrypto` method, call the `RemoveSecurity` method.

```
RemoveSecurity(_symbol);
```

The `RemoveSecurity` method cancels your open orders for the security and liquidates your holdings in the virtual pair .

Supported Assets

The following table shows the available Crypto Future pairs:

Pairs Available (631)					
1CATUSDT	1INCHUSDT	A8USDT	AAVEUSD	AAVEUSDT	ACEUSDT
ACHUSDT	ACTUSDT	ACXUSDT	ADAUSD	ADAUSDT	AERGOUSDT
AEROUSDT	AEVOUSDT	AGIUSDT	AGIXUSDT	AGLDUSDT	AI16ZUSDT
AIDOGEUSDT	AIOZUSDT	AIUSDT	AIXBTUSDT	AKROUSDT	AKTUSDT
ALCHUSDT	ALEOUSDT	ALGOUSDT	ALICEUSDT	ALPACAUSDT	ALPHAUSDT
ALTUSDT	ALUUSDT	AMBUSDT	ANIMEUSDT	ANKRUSDT	ANTUSDT
APEUSDT	APEXUSDT	API3USDT	APTUSD	APTUSDT	APUUSDT
ARBUSDT	ARCUSDT	ARKMUSDT	ARKUSDT	ARPAUSDT	ARUSDT
ASTRUSDT	ATAUSDT	ATHUSDT	ATOMUSDT	AUCTIONUSDT	AUDIOUSDT

AVAAIUSDT	AVAILUSDT	AVAUSDT	AVAXUSD	AVAXUSDT	AVLUSDT
AXLUSDT	AXSUSDT	B3USDT	BABYDOGEUSDT	BABYUSDT	BADGERUSDT
BAKEUSDT	BALUSDT	BANANAS31USDT	BANANAUSDT	BANDUSDT	BANKUSDT
BANUSDT	BATUSDT	BBUSDT	BCHUSD	BCHUSDT	BEAMUSDT
BEERUSDT	BELUSDT	BENDOGUSDT	BERAUSDT	BICOUSDT	BIGTIMEUSDT
BILLYUSDT	BIOUSDT	BLASTUSDT	BLUEUSDT	BLURUSDT	BLZUSDT
BMTUSDT	BNBUSDT	BNTUSDT	BNXUSDT	BOBAUSDT	BOMEUSDT
BONDUSDT	BONKUSDT	BRETTUSDT	BROCCOLIUSDT	BRUSDT	BSVUSDT
BSWUSDT	BTCUSD	BTCUSDT	BTTUSDT	BUSDUSDT	BUZZUSDT
C98USDT	CAKEUSDT	CARVUSDT	CATIUSDT	CATSUSDT	CATUSDT
CEEKUSDT	CELOUSDT	CELRUSDT	CETUSUSDT	CFXUSDT	CGPTUSDT
CHEEMSUSDT	CHESSUSDT	CHILLGUYUSDT	CHRUSDT	CHZUSDT	CKBUSDT
CLANKERUSDT	CLOUDUSDT	COMBOUSDT	COMPUSDT	COOKIEUSDT	COOKUSDT
COQUSDT	COREUSDT	COSUSDT	COTIUSDT	COVALUSDT	COWUSDT
CPOOLUSDT	CROUSDT	CRVUSDT	CTCUSDT	CTKUSDT	CTSIUSDT
CVCUSDT	CVXUSDT	CYBERUSDT	DAOUSDT	DARKUSDT	DARUSDT
DASHUSDT	DATAUSDT	DBRUSDT	DEEPUSDT	DEGENUSDT	DENTUSDT
DEXEUSDT	DGBUSDT	DODOUSDT	DOGEUSD	DOGEUSDT	DOGSUSDT
DOGUSDT	DOODUSDT	DOP1USDT	DOTUSD	DOTUSDT	DRIFTUSDT
DUCKUSDT	DUSKUSDT	DYDXUSDT	DYMUSDT	EDUUSDT	EGLDUSDT
EIGENUSDT	ELONUSDT	ELXUSDT	ENAUSDT	ENJUSDT	ENSUSDT
EOSUSD	EOSUSDT	EPICUSDT	EPTUSDT	ETCUSD	ETCUSDT
ETHBTCUSDT	ETHFIUSDT	ETHUSD	ETHUSDT	ETHWUSDT	FARTCOINUSDT
FBUSDT	FDUSDUSDT	FETUSDT	FHEUSDT	FIDAUSDT	FILUSD
FILUSDT	FIOUSDT	FIREUSDT	FITFIUSDT	FLMUSDT	FLOCKUSDT
FLOKIUSDT	FLOWUSDT	FLRUSDT	FLUXUSDT	FORMUSDT	FORTHUSDT
FOXYUSDT	FRONTUSDT	FTMUSDT	FTNUSDT	FUELUSDT	FUNUSDT
FUSDT	FWOGUSDT	FXSUSDT	GALAUSDT	GALUSDT	GASUSDT

GEMSUSDT	GFTUSDT	GIGAUSDT	GLMRUSDT	GLMUSDT	GMEUSDT
GMTUSDT	GMXUSDT	GNOUSDT	GOATUSDT	GODSUSDT	GOMININGUSDT
GORKUSDT	GPSUSDT	GPTUSDT	GRASSUSDT	GRIFFAINUSDT	GRTUSDT
GTCUSDT	GUNUSDT	GUSDT	HAEDALUSDT	HBARUSDT	HEIUSDT
HFTUSDT	HIFIUSDT	HIGHUSDT	HIPPOUSDT	HIVEUSDT	HMSTRUSDT
HNTUSDT	HOOKUSDT	HOTUSDT	HPOS10IUSDT	HYPERUSDT	HYPEUSDT
ICPUSDT	ICXUSDT	IDEXUSDT	IDUSDT	ILVUSDT	IMXUSDT
INITUSDT	INJUSDT	IOSTUSDT	IOTAUSDT	IOTXUSDT	IOUSDT
IPUSDT	IQ50USDT	JASMYUSDT	JELLYJELLYUSDT	JOEUSDT	JSTUSDT
JTOUSDT	JUPUSDT	JUSDT	KAIAUSDT	KAITOUSDT	KASUSDT
KAVAUSDT	KDAUSDT	KERNELUSDT	KEYUSDT	KLAYUSDT	KMNOUSDT
KNCUSDT	KOMAUSDT	KSMUSDT	L3USDT	LADYSUSDT	LAIUSDT
LDOUSDT	LEVERUSDT	LINAUSDT	LINKUSD	LINKUSDT	LISTAUSDT
LITUSDT	LOOKSUSDT	LOOMUSDT	LPTUSDT	LQTYUSDT	LRCUSDT
LSKUSDT	LTCUSD	LTCUSDT	LTOUSDT	LUCEUSDT	LUMIAUSDT
LUNA2USDT	LUNCUSDT	MAGICUSDT	MAJORUSDT	MANAUSD	MANAUSDT
MANEKIUSDT	MANTAUSDT	MAPOUSDT	MASAUSDT	MASKUSDT	MATICUSDT
MAVIAUSDT	MAVUSDT	MAXUSDT	MBLUSDT	MBOXUSDT	MCUSDT
MDTUSDT	MELANIAUSDT	MEMEFIUSDT	MEMEUSDT	MERLUSDT	METISUSDT
MEUSDT	MEWUSDT	MICHIUSDT	MILKUSDT	MINAUSDT	MKRUSDT
MLNUSDT	MNTUSD	MNTUSDT	MOBILEUSDT	MOCAUSDT	MOGUSDT
MONUSDT	MOODENGUSDT	MORPHOUSDT	MOTHER1USDT	MOTHER2USDT	MOTHERUSDT
MOVEUSDT	MOVRUSDT	MTLUSDT	MUBARAKUSDT	MULTIUSDT	MUMUUSDT
MVLUSDT	MYRIAUSDT	MYROUSDT	NCUSDT	NEARUSD	NEARUSDT
NEIROCTOUSDT	NEIROETHUSDT	NEOUSDT	NFPUSDT	NFTUSDT	NILUSDT
NKNUSDT	NMRUSDT	NOTUSDT	NSUSDT	NTRNUSDT	NULSUSDT
NYANUSDT	OBOLUSDT	OBTUSDT	OCEANUSDT	OGNUSDT	OGUSDT
OKBUSDT	OLUSDT	OMGUSDT	OMNIUSDT	OMUSDT	ONDOUSDT

ONEUSDT	ONGUSDT	ONTUSDT	OPUSD	OPUSDT	ORBSUSDT
ORCAUSDT	ORDERUSDT	ORDIUSDT	ORNUSDT	OSMOUSDT	OXTUSDT
PARTIUSDT	PAXGUSDT	PEAQUSDT	PEIPEIUSDT	PENDLEUSDT	PENGUSDT
PENGUUSDT	PEOPLEUSDT	PEPEUSDT	PERPUSDT	PHAUSDT	PHBUSDT
PIPPINUSDT	PIRATEUSDT	PIXELUSDT	PIXFIUSDT	PLUMEUSDT	PNUTUSDT
POLUSDT	POLYXUSDT	PONKEUSDT	POPCATUSDT	PORTALUSDT	POWRUSDT
PRCLUSDT	PRE05EOSUSDT	PRE08EOSUSDT	PRIMEUSDT	PROMPTUSDT	PROMUSDT
PROSUSDT	PUFFERUSDT	PUMPUSDT	PUNDIXUSDT	PUNDUUSDT	PYRUSDT
PYTHUSDT	QIUSDT	QNTUSDT	QTUMUSDT	QUBICUSDT	QUICKUSDT
RADUSDT	RAREUSDT	RATSUSDT	RAYDIUMUSDT	RAYUSDT	RDNTUSDT
REDUSDT	REEFUSDT	RENDERUSDT	RENUSDT	REQUSDT	REXUSDT
REZUSDT	RFCUSDT	RIFSOLUSDT	RIFUSDT	RLCUSDT	RNDRUSDT
ROAMUSDT	RONINUSDT	RONUSDT	ROSEUSDT	RPLUSDT	RSRUSDT
RSS3USDT	RUNEUSDT	RVNUSDT	SAFEUSDT	SAGAUSDT	SANDUSDT
SATSUSDT	SCAUSDT	SCRTUSDT	SCRUSDT	SCUSDT	SDUSDT
SEIUSDT	SENDUSDT	SERAPHUSDT	SFPUSDT	SHELLUSDT	SHIB1000USDT
SIGNUSDT	SILLYUSDT	SIRENUSDT	SKLUSDT	SLERFUSDT	SLFUSDT
SLPUSDT	SNTUSDT	SNXUSDT	SOLAYERUSDT	SOLOUSDT	SOLUSD
SOLUSDT	SOLVUSDT	SONICUSDT	SPECUSDT	SPELLUSDT	SPXUSDT
SSVUSDT	STARLUSDT	STEEMUSDT	STGUSDT	STMXUSDT	STORJUSDT
STOUSDT	STPTUSDT	STRAXUSDT	STRKUSDT	STXUSDT	SUIUSD
SUIUSDT	SUNDOGUSDT	SUNUSDT	SUPERUSDT	SUSDT	SUSHIUSDT
SWARMSUSDT	SWEATUSDT	SWELLUSDT	SXPUSDT	SXTUSDT	SYNUSDT
SYSUSDT	TAIKOUSDT	TAIUSDT	TAOUSDT	TEST01SWELLUSDT	THETAUSDT
THEUSDT	TIAUSDT	TLMUSDT	TNSRUSDT	TOKENUSDT	TOMIUSDT
TOMOUSDT	TONUSD	TONUSDT	TOSHIUSDT	TRBUSDT	TROYUSDT
TRUMPUSDT	TRUUSDT	TRXUSDT	TSTBSCUSDT	TURBOUSDT	TUSDT
TUTUSDT	TWTUSDT	UMAUSDT	UNFIUSDT	UNIUSD	UNIUSDT

UROUSDT	USDCUSDT	USDEUSDT	USTCUSDT	USUALUSDT	UXLINKUSDT
VANAUSDT	VANRYUSDT	VELODROMEUSDT	VELOUSDT	VETUSDT	VGXUSDT
VICUSDT	VIDTUSDT	VINEUSDT	VINUUSDT	VIRTUALUSDT	VOXELUSDT
VRAUSDT	VRUSDT	VTHOUSDT	VVVUSDT	WALUSDT	WAVESUSDT
WAXPUSDT	WCTUSDT	WENUSDT	WHYUSDT	WIFUSD	WIFUSDT
WLDUSDT	WOOUSDT	WSMUSDT	WUSDT	XAIUSDT	XAUTUSDT
XCHUSDT	XCNUSDT	XDCUSDT	XECUSDT	XEMUSDT	XIONUSDT
XLMUSD	XLMUSDT	XMRUSDT	XNOUSDT	XRDUSDT	XRPUSD
XRPUSDT	XTERUSDT	XTZUSDT	XUSDT	XVGUSDT	XVSUSDT
YFIIUSDT	YFIUSDT	YGGUSDT	ZBCNUSDT	ZCXUSDT	ZECUSDT
ZENTUSDT	ZENUSDT	ZEREBROUSDT	ZETAUSDT	ZEUSUSDT	ZILUSDT
ZKFUSDT	ZKJUSDT	ZKUSDT	ZORAUSDT	ZRCUSDT	ZROUSDT
ZRXUSDT					

Example Applications

The Binance Crypto Future Margin Rate dataset enables correct margin cost so you can accurately design strategies for Cryptocurrencies with term structure. Examples include the following strategies:

- Horizontal/Diagonal arbitrage with the underlying Cryptocurrencies
- Trade Contango/Backwardation predictions
- Hedge for illiquid Cryptocurrencies

Classic Algorithm Example

The following example algorithm holds a long position in the BTCUSDT perpetual Future contract on the Bybit exchange. When the margin interest rate is relatively low, it flips to a short position.

```C#
public class BybitCryptoFutureDataAlgorithm : QCAlgorithm
{
    private CryptoFuture _future;
    // Add an indicator to track the low of trailing margin interest rates.
    private Minimum _lowInterestRate = new(10);

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Set Account Currency to USDT, since USD cannot be used to trade.
        SetCash("USDT", 100000);
        // Bybit  accepts both Cash and Margin account types, select the one you need for the best reality modeling.
        SetBrokerageModel(BrokerageName.Bybit, AccountType.Margin);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Requesting data, we only trade on BTCUSDT Future in Bybit  exchange
        // Perpetual futures does not have a filter function.
        _future = AddCryptoFuture("BTCUSDT", Resolution.Daily);
        // Historical data
        var history = History<MarginInterestRate>(_future.Symbol, 10, Resolution.Daily);
        foreach (var dataPoint in history)
        {
            _lowInterestRate.Update(dataPoint.EndTime, dataPoint.InterestRate);
        }
    }

    public override void OnData(Slice slice)
    {
        // Note that you may want to access the margin interest of the crypto future to calculate if it would impact
a trade's PnL
        // Or you can calculate the trade size on keeping the quote currency constant
        if (!slice.MarginInterestRates.TryGetValue(_future.Symbol, out var dataPoint))
        {
            return;
        }
        var interestRate = dataPoint.InterestRate;
        Plot("Margin Interest Rate", _future.Symbol.ToString(), interestRate);
        // Short when margin interest rates are relatively low. Otherwise, long.
        if (interestRate <= _lowInterestRate)
        {
            if (!_future.Holdings.IsShort)
            {
                SetHoldings(_future.Symbol, -0.5);
            }
        }
        else if (!_future.Holdings.IsLong)
        {
            SetHoldings(_future.Symbol, 1);
        }
        // Update the indicator.
        _lowInterestRate.Update(Time, interestRate);
    }
}
```

Framework Algorithm Example

The following example algorithm holds a long position in the BTCUSDT perpetual Future contract on the Bybit exchange. When the margin interest rate is relatively low, it flips to a short position.

```C#
public class BybitCryptoFutureDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        Settings.FreePortfolioValuePercentage = 0.1m;
        // Set Account Currency to USDT, since USD cannot be used to trade.
        SetCash("USDT", 100000);
        // Bybit accepts both Cash and Margin account types, select the one you need for the best reality modeling.
        SetBrokerageModel(BrokerageName.Bybit, AccountType.Margin);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        UniverseSettings.Resolution = Resolution.Daily;
```

```
        UniverseSettings.Leverage = 2;
        // We only trade on BTCUSDT Future in Bybit exchange
        var symbols = new List<Symbol>{
            QuantConnect.Symbol.Create("BTCUSDT", SecurityType.CryptoFuture, Market.Bybit)
        };
        AddUniverseSelection(new ManualUniverseSelectionModel(symbols));
        // Custom alpha model to emit insights based on the Crypto Future price data
        AddAlpha(new CryptoFutureAlphaModel());
        // Equally invest to evenly dissipate the capital concentration risk of inidividual crypto pair
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
        SetExecution(new ImmediateExecutionModel());
    }
}

public class CryptoFutureAlphaModel : AlphaModel
{
    private Dictionary<Symbol, Minimum> _minBySymbol = new();

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = new List<Insight>();
        foreach (var kvp in _minBySymbol)
        {
            var symbol = kvp.Key;
            var lowInterestRate = kvp.Value;
            // Note that you may want to access the margin interest of the crypto future to calculate if it would
impact a trade's PnL
            // Or you can calculate the trade size on keeping the quote currency constant
            if (!slice.MarginInterestRates.TryGetValue(symbol, out var dataPoint))
                continue;
            var interestRate = dataPoint.InterestRate;
            algorithm.Plot("Margin Interest Rate", symbol.ToString(), interestRate);
            // Short when margin interest rates are relatively low. Otherwise, long.
            var direction = interestRate <= lowInterestRate.Current.Value
                ? InsightDirection.Down
                : InsightDirection.Up;
            insights.Add(Insight.Price(symbol, TimeSpan.FromDays(1), direction));
            // Update the indicator.
            lowInterestRate.Update(slice.Time, interestRate);
        }
        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            var symbol = security.Symbol;
            // Add an indicator to track the low of trailing margin interest rates.
            var min = new Minimum(10);
            _minBySymbol[symbol] = min;
            // Historical data
            var history = algorithm.History<MarginInterestRate>(symbol, 10, Resolution.Daily);
            foreach (var dataPoint in history)
            {
                min.Update(dataPoint.EndTime, dataPoint.InterestRate);
            }
        }
        foreach (var security in changes.RemovedSecurities)
        {
            _minBySymbol.Remove(security.Symbol);
        }
    }
}
```

Data Point Attributes

The Binance Crypto Future Margin Rate dataset provides `MarginInterestRate` objects, which have the following attributes:

QuantConnect

CFD Data

Introduction

The CFD Data by QuantConnect serves 51 contracts for differences (CFD). The data starts as early as May 2002 and is delivered on any frequency from tick to daily. This dataset is created by QuantConnect processing raw tick data from OANDA.

CFD data does not include ask and bid sizes.

For more information about the CFD Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Getting Started

The following snippet demonstrates how to request data from the CFD dataset:

```C#
_symbol = AddCfd("XAUUSD", Resolution.Daily).Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	Mixed, earliest starts May 2002
Asset Coverage	51 Contracts
Data Density	Dense
Resolution	Tick, Second, Minute, Hour, & Daily
Timezone	Mixed, in which the contract is listed*
Market Hours	Always Open , except from Friday 5 PM EST to Sunday 5 PM EST. Index CFDs depends on the underlying market hour*

* E.g.: DE30EUR tracks DAX30 Index, which is listed in Europe/Berlin timezone.

Requesting Data

To add CFD data to your algorithm, call the `AddCfd` method. Save a reference to the CFD `Symbol` so you can access the data later in your algorithm.

```
C#
public class CfdAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetAccountCurrency("EUR");

        SetStartDate(2019, 2, 20);
        SetEndDate(2019, 2, 21);
        SetCash("EUR", 100000);

        _symbol = AddCfd("DE30EUR").Symbol;

        SetBenchmark(_symbol);
    }
}
```

For more information about creating CFD subscriptions, see Requesting Data .

Accessing Data

To get the current CFD data, index the `QuoteBars` , or `Ticks` properties of the current `Slice` with the CFD `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.QuoteBars.ContainsKey(_symbol))
    {
        var quoteBar = slice.QuoteBars[_symbol];
        Log($"{_symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    if (slice.Ticks.ContainsKey(_symbol))
    {
        var ticks = slice.Ticks[_symbol];
        foreach (var tick in ticks)
        {
            Log($"{_symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

You can also iterate through all of the data objects in the current `Slice` .

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        Log($"{symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    foreach (var kvp in slice.Ticks)
    {
        var symbol = kvp.Key;
        var ticks = kvp.Value;
        foreach (var tick in ticks)
        {
            Log($"{symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

For more information about accessing CFD data, see Handling Data .

Historical Data

To get historical CFD data, call the `History` method with the CFD `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
// QuoteBar objects
var historyQuoteBars = History<QuoteBar>(_symbol, 100, Resolution.Minute);

// Tick objects
var historyTicks = History<Tick>(_symbol, TimeSpan.FromSeconds(10), Resolution.Tick);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a CFD subscription, call the `RemoveSecurity` method.

```
C#
RemoveSecurity(_symbol);
```

The `RemoveSecurity` method cancels your open orders for the security and liquidates your holdings.

Supported Assets

The following table shows the available contracts:

Contract Available (51)					
AU200AUD	BCOUSD	CH20CHF	CORNUSD	DE10YBEUR	DE30EUR
EU50EUR	FR40EUR	HK33HKD	JP225USD	NAS100USD	NATGASUSD
NL25EUR	SG30SGD	SOYBNUSD	SPX500USD	SUGARUSD	UK100GBP
UK10YBGBP	US2000USD	US30USD	USB02YUSD	USB05YUSD	USB10YUSD
USB30YUSD	WHEATUSD	WTICOUSD	XAGAUD	XAGCAD	XAGCHF
XAGEUR	XAGGBP	XAGHKD	XAGJPY	XAGNZD	XAGSGD
XAGUSD	XAUAUD	XAUCAD	XAUCHF	XAUEUR	XAUGBP
XAUHKD	XAUJPY	XAUNZD	XAUSGD	XAUUSD	XAUXAG
XCUUSD	XPDUSD	XPTUSD			

Example Applications

The CFD price data enables you to trade CFD assets in your algorithm. Examples include the following strategies:

- Exploring the daily worldwide news cycles with CFDs that track international indices.
- Trading price movements of commodities with no delivery of physical goods. For example, pairs trading between gold and silver, corn and wheat, brent and crude oil, etc.

Classic Algorithm Example

The following example algorithm implements a pairs trading strategy using Gold and Silver CFDs, XAUUSD and XAGUSD, respectively. When the spread is higher than one standard deviation above its mean, the algorithm buys the spread (buy XAUUSD and sell XAGUSD). When the spread is lower than one standard deviation below its mean, it sells the spread (buy XAGUSD and sell XAUUSD).

C#

```
public class GoldSilverPairsTradingAlgorithm : QCAlgorithm
{
    // Use 500-step mean and SD indicator on determine the spread relative difference for trading signal generation
    private SimpleMovingAverage _spreadMean = new SimpleMovingAverage(500);
    private StandardDeviation _spreadStd = new StandardDeviation(500);
    private Security[] _pair = new Security[2];

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Request gold and sliver spot CFDs for trading their spread difference, assuming their spread series is
cointegrated
        AddCfd("XAUUSD", Resolution.Hour);
        AddCfd("XAGUSD", Resolution.Hour);
    }

    public override void OnData(Slice slice)
    {
        // Update the indicator with updated spread difference, such that the an updated cointegration threshold is
calculated for trade inception
        var spread = _pair[1].Price - _pair[0].Price;
        _spreadMean.Update(Time, spread);
        _spreadStd.Update(Time, spread);

        var upperthreshold = _spreadMean + _spreadStd;
        var lowerthreshold = _spreadMean - _spreadStd;

        // If the spread is higher than upper threshold, bet their spread series will revert to mean
        if (spread > upperthreshold)
        {
            SetHoldings(_pair[0].Symbol, 1);
            SetHoldings(_pair[1].Symbol, -1);
        }
        else if (spread < lowerthreshold)
        {
            SetHoldings(_pair[0].Symbol, -1);
            SetHoldings(_pair[1].Symbol, 1);
        }
        // Close positions if mean reverted
        else if ((Portfolio[_pair[0].Symbol].Quantity > 0m && spread < _spreadMean)
        || (Portfolio[_pair[0].Symbol].Quantity < 0m && spread > _spreadMean))
        {
            Liquidate();
        }
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        _pair = changes.AddedSecurities.ToArray();

        //1. Call for 500 days of history data for each symbol in the pair and save to the variable history
        var history = History(_pair.Select(x => x.Symbol), 500);

        //2. Iterate through the history tuple and update the mean and standard deviation with historical data
        foreach(var slice in history)
        {
            var spread = slice[_pair[1].Symbol].Close - slice[_pair[0].Symbol].Close;
            _spreadMean.Update(slice.Time, spread);
            _spreadStd.Update(slice.Time, spread);
        }
    }
}
```

Framework Algorithm Example

The following example algorithm implements a pairs trading strategy using Gold and Silver CFDs, XAUUSD and XAGUSD, respectively. When the spread is higher than one standard deviation above its mean, the algorithm buys the spread (buy XAUUSD and sell XAGUSD). When the spread is lower than one standard deviation below its mean, it sells the spread (buy XAGUSD and sell XAUUSD).

C#

```
public class GoldSilverPairsTradingAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        UniverseSettings.Resolution = Resolution.Hour;
        // Custom universe contains only gold and sliver spot CFDs for trading their spread difference, assuming
their spread series is cointegrated
        SetUniverseSelection(new ManualUniverseSelectionModel
        (
            new [] {"XAUUSD", "XAGUSD"}
                .Select(x => QuantConnect.Symbol.Create(x, SecurityType.Cfd, Market.Oanda))
        ));
        // Custom alpha model to emit trade insights based on the gold-sliver price spread
        AddAlpha(new PairsTradingAlphaModel());
        // Equal weighting trades assuming the spread is cointegrated by 1:1 ratio
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }
}

public partial class PairsTradingAlphaModel : AlphaModel
{
    // Use 500-step mean and SD indicator on determine the spread relative difference for trading signal generation
    private SimpleMovingAverage _spreadMean = new SimpleMovingAverage(500);
    private StandardDeviation _spreadStd = new StandardDeviation(500);
    // Assume efficient mean reversal happens within 2 hours
    private TimeSpan _period = TimeSpan.FromHours(2);
    private Security[] _pair = new Security[2];

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        // Update the indicator with updated spread difference, such that the an updated cointegration threshold is
calculated for trade inception
        var spread = _pair[1].Price - _pair[0].Price;
        _spreadMean.Update(algorithm.Time, spread);
        _spreadStd.Update(algorithm.Time, spread);

        var upperthreshold = _spreadMean + _spreadStd;
        var lowerthreshold = _spreadMean - _spreadStd;

        // If the spread is higher than upper threshold, bet their spread series will revert to mean
        if (spread > upperthreshold)
        {
            return Insight.Group(
                Insight.Price(_pair[0].Symbol, _period, InsightDirection.Up),
                Insight.Price(_pair[1].Symbol, _period, InsightDirection.Down)
            );
        }
        else if (spread < lowerthreshold)
        {
            return Insight.Group(
                Insight.Price(_pair[0].Symbol, _period, InsightDirection.Down),
                Insight.Price(_pair[1].Symbol, _period, InsightDirection.Up)
            );
        }

        return Enumerable.Empty<Insight>();
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        _pair = changes.AddedSecurities.ToArray();

        //1. Call for 500 days of history data for each symbol in the pair and save to the variable history
        var history = algorithm.History(_pair.Select(x => x.Symbol), 500);
```

```
        //2. Iterate through the history tuple and update the mean and standard deviation with historical data
        foreach (var slice in history)
        {
            var spread = slice[_pair[1].Symbol].Close - slice[_pair[0].Symbol].Close;
            _spreadMean.Update(slice.Time, spread);
            _spreadStd.Update(slice.Time, spread);
        }
    }
}
```

Data Point Attributes

The CFD dataset provides `QuoteBar` and `Tick` objects.

QuoteBar Attributes

`QuoteBar` objects have the following attributes:

Tick Attributes

`Tick` objects have the following attributes:

QuantConnect

Cash Indices

Introduction

The Cash Indices dataset by QuantConnect covers 125 US Indices and 3 International indices. The data starts on various dates from January 1998 and is delivered on any frequency from minute to daily.

For more information about the Cash Indices dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Live Data

The Cash Indices dataset does not provide a live data feed. To receive Cash Indices' data in your live algorithm, you must add a brokerage or a third-party data provider (see Cloud Platform > Datasets).

Getting Started

The following snippet demonstrates how to request data from the Cash Indices dataset:

```
C#
_symbol = AddIndex("VIX", Resolution.Daily).Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 1998
Coverage	125 US Indices and 3 International indices (HSI, SX5E and N225)
Data Density	Dense
Resolution	Minute, Hour, & Daily
Timezone	New York for US Indices, refer to `Supported Assets` below for International indices
Market Hours	Regular Only

Requesting Data

To add Cash Indices data to your algorithm, call the `AddIndex` method. Save a reference to the Index `Symbol` so you can access the data later in your algorithm.

```C#
public class CashIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2020, 6, 1);
        SetEndDate(2021, 6, 1);
        SetCash(100000);

        _symbol = AddIndex("VIX").Symbol;
    }
}
```

For more information about creating Index subscriptions, see Requesting Data .

Accessing Data

To get the current Cash Indices data, index the `Bars` property of the current `Slice` with the Index `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```C#
public override void OnData(Slice slice)
{
    if (slice.Bars.ContainsKey(_symbol))
    {
        var tradeBar = slice.Bars[_symbol];
        Log($"{_symbol} price at {slice.Time}: {tradeBar.Close}");
    }
}
```

You can also iterate through all of the data objects in the current `Slice` .

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Bars)
    {
        var symbol = kvp.Key;
        var tradeBar = kvp.Value;
        Log($"{symbol} price at {slice.Time}: {tradeBar.Close}");
    }
}
```

For more information about accessing Index data, see Handling Data .

Historical Data

To get historical Cash Indices data, call the `History` method with the Index `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
// TradeBar objects
var historyBars = History(_symbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove an Index subscription, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_symbol);
```

Supported Indices

US Cash Indices

The following table shows the US Cash indices with Index Option support:

Ticker	Index	Expiry	Start Date
VIX	S&P500	30 Days	Jul 2003
SPX	S&P500	-	Jan 1998
NDX	NASDAQ-100	-	Jan 1998
RUT	Russell 2000	-	Jan 2008

VIX - CBOE Volatility Index

The Cboe Volatility Index (VIX) is a real-time index that represents the market's expectations for the relative strength of near-term price changes of the S&P 500 Index (SPX). Because it's derived from the prices of SPX index Options with near-term expiration dates, it generates a 30-day forward projection of volatility. Volatility, or how fast prices change, is often seen as a way to gauge market sentiment, and in particular, the degree of fear among market participants.

SPX - S&P 500 Index

The S&P 500 Index, or the Standard & Poor's 500 Index, is a market-capitalization-weighted index of the 500 largest publicly-traded companies in the U.S. It is not an exact list of the top 500 U.S. companies by market capitalization because there are other criteria included in the index. The index is widely regarded as the best gauge of large-cap U.S. Equities.

NDX - Nasdaq 100 Index

The Nasdaq-100 Index is a modified market-capitalization-weighted index composed of securities issued by 100 of the largest non-financial companies listed on the Nasdaq Stock Market (Nasdaq). The index includes companies from various industries except for the financial industry, like commercial and investment banks. These non-financial sectors include retail, biotechnology, industrial, technology, health care, and others.

RUT - Russell 2000 Index

The Russell 2000 Index is a small-cap U.S. stock market index that makes up the smallest 2,000 stocks in the Russell Index. It was started by the Frank Russell Company in 1984. The index is maintained by FTSE Russell, a subsidiary of the London Stock Exchange Group (LSEG).

All US Cash Indices

The following table shows all US Cash indices:

Indices Available (125). Note: QuantConnect is working on its modeling: market hours and symbol properties.					
AEX	ASX	BKX	BRR	BRTI	BXD
BXM	BXN	BXR	CEX	CLL	COMP
COR1M	COR1Y	COR30D	COR3M	COR6M	COR9M
CXU	DAX	DJCIAGC	DJCICC	DJCIEN	DJCIGC
DJCIGR	DJCIIK	DJCIKC	DJCISB	DJCISI	DJI
DJINET	DJR	DJTTR	DJX	DRG	DUX
DVS	DWCF	DWCPF	DWLG	DWLV	DXL
DXY	EVZ	FVX	GVZ	HGX	LOVOL
MID	MIDG	MIDV	MRUT	NDX	NQX
NYA	NYFANG	NYXBT	OEX	OSX	OVX
PUT	REIT	RUA	RUI	RUT	RVX
RXM	SET	SGX	SKEW	SMILE	SMLG
SP500LVOL	SP600	SPEN	SPGSCI	SPMPG	SPMPV
SPRI	SPRO	SPSIBI	SPSPG	SPSPV	SPSV
SPTMI	SPX	SPXPG	SPXPV	SVX	TNX
TYX	UKX	UTIL	UTY	VIF	VIN
VIX	VIX1D	VIX1Y	VIX3M	VIX6M	VIX9D
VOLI	VPD	VPN	VVIX	VWA	VWB
VXD	VXN	VXO	VXSLV	VXTH	VXTLT
W5000	WPUT	XAU	XAX	XDA	XDB
XEO	XMI	XNDX	XSP	XSR	

International Cash Indices

The following table shows the International Cash indices:

Ticker	Index	Start Date	Time Zone	Currency
HSI	Hang Seng Index	Dec 2006	Asia/Hong Kong	HKD
SX5E	EUREX EU STOXX	Jul 2003	Europe/Berlin	EUR
N225	Nikkei 225	Jul 2003	Asia/Tokyo	JPY

Example Applications

The Cash Indices enable you to incorporate popular indices into your trading algorithms. Examples include the following use cases:

- Exploring the difference between the Index and the ETF that tracks it
- Using these indices as the underlying asset fo Index Options strategies
- Understanding the stock market's level of expected forward-looking volatility, also known as the "fear index". When the VIX starts moving higher, it tells you that traders are getting nervous. When the VIX starts moving lower, it tells you that traders are gaining confidence.

Classic Algorithm Example

The following example algorithm tracks the 80-day EMA and 200-day EMA of SPX. When the short EMA crosses above the long EMA, the algorithm buys SPY. Otherwise, it holds cash.

```C#
public class IndexDataAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private ExponentialMovingAverage _emaSlow, _emaFast;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);

        // Request SPY data as trading vehicle for SPX
        _spy = AddEquity("SPY").Symbol;

        // Request SPX data for trade signal generation
        var spx = AddIndex("SPX").Symbol;

        // Create short and long-term EMA indicators for trend estimation to trade
        _emaFast = EMA(spx, 80, Resolution.Daily);
        _emaSlow = EMA(spx, 200, Resolution.Daily);
        SetWarmUp(200, Resolution.Daily);

        // Historical data
        var history = History(spx, 60, Resolution.Daily);
        Debug($"We got {history.Count()} items from our history request");
    }

    public override void OnData(Slice slice)
    {
        // Trade signals required indicators to be ready
        if (IsWarmingUp || !_emaSlow.IsReady)
        {
            return;
        }

        // If short-term EMA is above long-term, it indicates an up trend, so we buy SPY
        if (!Portfolio.Invested && _emaFast > _emaSlow)
        {
            SetHoldings(_spy, 1);
        }
        // If it is the reverse, it indicates a downtrend, and we liquidate any position
        else if (_emaFast < _emaSlow)
        {
            Liquidate();
        }
    }
}
```

Framework Algorithm Example

The following example algorithm tracks the 80-day EMA and 200-day EMA of SPX. When the short EMA crosses above the long EMA, the algorithm buys SPY. Otherwise, it holds cash.

C#

```
public class IndexDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);

        // Universe contains only SPY as a trading vehicle for SPX
        SetUniverseSelection(new ManualUniverseSelectionModel
        (
            QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA)
        ));
        // Custom alpha model that emits insights based on SPX index data
        SetAlpha(new SpxEmaCrossAlphaModel(this));
        // Equally invests can dissipate non-systematic risky event's capital concentration risk evenly
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel(Expiry.EndOfMonth));
    }
}

public class SpxEmaCrossAlphaModel : AlphaModel
{
    private ExponentialMovingAverage _emaSlow, _emaFast;

    public SpxEmaCrossAlphaModel(QCAlgorithm algorithm)
    {
        // Request SPX data for trade signal generation
        var spx = algorithm.AddIndex("SPX").Symbol;

        // Create short and long-term EMA indicators for trend estimation to trade
        _emaFast = algorithm.EMA(spx, 80, Resolution.Daily);
        _emaSlow = algorithm.EMA(spx, 200, Resolution.Daily);
        algorithm.SetWarmUp(200, Resolution.Daily);

        // Historical data
        var history = algorithm.History(spx, 60, Resolution.Daily);
        algorithm.Debug($"We got {history.Count()} items from our history request");
    }

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        // Trade signals required indicators to be ready
        if (algorithm.IsWarmingUp || !_emaSlow.IsReady)
        {
            return Enumerable.Empty<Insight>();
        }

        // If short-term EMA is above long-term, it indicates an up trend, so we buy SPY
        if (!algorithm.Portfolio.Invested && _emaFast > _emaSlow)
        {
            return algorithm.ActiveSecurities
                .Where(kvp => kvp.Value.IsTradable)
                .Select(kvp => Insight.Price(kvp.Key, Expiry.EndOfMonth, InsightDirection.Up));
        }
        // If it is the reverse, it indicates downtrend, and we liquidate any position
        else if (_emaFast < _emaSlow)
        {
            return algorithm.ActiveSecurities
                .Where(kvp => kvp.Value.IsTradable)
                .Select(kvp => Insight.Price(kvp.Key, Expiry.EndOfMonth, InsightDirection.Flat));
        }

        return Enumerable.Empty<Insight>();
    }
}
```

Data Point Attributes

The Cash Indices dataset provides `TradeBar` object.

TradeBar Attributes

`TradeBar` objects have the following attributes:

QuantConnect

FOREX Data

Introduction

The FOREX Data by QuantConnect serves 71 foreign exchange (FOREX) pairs, starts on various dates from January 2007, and is delivered on any frequency from tick to daily. This dataset is created by QuantConnect processing raw tick data from OANDA.

FOREX data does not include ask and bid sizes.

For more information about the FOREX Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Getting Started

The following snippet demonstrates how to request data from the FOREX dataset:

```
C#
_symbol = AddForex("EURUSD", Resolution.Daily).Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2007
Asset Coverage	71 Currency pairs
Data Density	Dense
Resolution	Tick, Second, Minute, Hour, & Daily
Timezone	UTC
Market Hours	Always Open , except from Friday 5 PM EST to Sunday 5 PM EST

Requesting Data

To add FOREX data to your algorithm, call the `AddForex` method. Save a reference to the Forex `Symbol` so you can access the data later in your algorithm.

```
C#
public class ForexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2019, 2, 20);
        SetEndDate(2019, 2, 21);
        SetCash(100000);

        _symbol = AddForex("EURUSD", Resolution.Minute).Symbol;

        SetBenchmark(_symbol);
    }
}
```

For more information about creating Forex subscriptions, see Requesting Data .

Accessing Data

To get the current Forex data, index the `QuoteBars` , or `Ticks` properties of the current `Slice` with the Forex `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.QuoteBars.ContainsKey(_symbol))
    {
        var quoteBar = slice.QuoteBars[_symbol];
        Log($"{_symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    if (slice.Ticks.ContainsKey(_symbol))
    {
        var ticks = slice.Ticks[_symbol];
        foreach (var tick in ticks)
        {
            Log($"{_symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

You can also iterate through all of the data objects in the current `Slice` .

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        Log($"{symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    foreach (var kvp in slice.Ticks)
    {
        var symbol = kvp.Key;
        var ticks = kvp.Value;
        foreach (var tick in ticks)
        {
            Log($"{symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

For more information about accessing Forex data, see Handling Data .

Historical Data

To get historical Forex data, call the `History` method with the Forex `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#

// QuoteBar objects
var historyQuoteBars = History<QuoteBar>(_symbol, 100, Resolution.Minute);

// Tick objects
var historyTicks = History<Tick>(_symbol, TimeSpan.FromSeconds(10), Resolution.Tick);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a Forex pair subscription, call the `RemoveSecurity` method.

```
C#

RemoveSecurity(_symbol);
```

The `RemoveSecurity` method cancels your open orders for the security and liquidates your holdings.

Supported Assets

The following table shows the available Forex pairs:

Pairs Available (71)					
AUDCAD	AUDCHF	AUDHKD	AUDJPY	AUDNZD	AUDSGD
AUDUSD	CADCHF	CADHKD	CADJPY	CADSGD	CHFHKD
CHFJPY	CHFZAR	EURAUD	EURCAD	EURCHF	EURCZK
EURDKK	EURGBP	EURHKD	EURHUF	EURJPY	EURNOK
EURNZD	EURPLN	EURSEK	EURSGD	EURTRY	EURUSD
EURZAR	GBPAUD	GBPCAD	GBPCHF	GBPHKD	GBPJPY
GBPNZD	GBPPLN	GBPSGD	GBPUSD	GBPZAR	HKDJPY
NZDCAD	NZDCHF	NZDHKD	NZDJPY	NZDSGD	NZDUSD
SGDCHF	SGDHKD	SGDJPY	TRYJPY	USDCAD	USDCHF
USDCNH	USDCZK	USDDKK	USDHKD	USDHUF	USDINR
USDJPY	USDMXN	USDNOK	USDPLN	USDSAR	USDSEK
USDSGD	USDTHB	USDTRY	USDZAR	ZARJPY	

Example Applications

The FOREX price data enables you to trade currency pairs in the global market. Examples include the following strategies:

- Exploring the impact that daily worldwide news cycles have on international currencies
- Carry Trade: borrow from a lower interest currency pair to fund the purchase of a currency pair with a higher interest rate

Classic Algorithm Example

The following example algorithm implements a FOREX carry trade. It buys the FOREX pair of the country with the lowest interest rate and sells the FOREX pair of the country with the highest interest rate.

```C#
public class ForexCarryTradeAlgorithm : QCAlgorithm
{
    // Use a month counter as variable to control rebalancing
    private int _month = -1;
    private Dictionary<string, decimal> _rates;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(25000);

        // We will use hard-coded interest rates for each base currency
        _rates = new Dictionary<string, decimal>()
        {
            {"USDAUD", 1.5m},    // Australia
            {"USDCAD", 0.5m},    // Canada
            {"USDCNY", 4.35m},   // China
            {"USDEUR", 0.0m},    // Euro Area
            {"USDINR", 6.5m},    // India
            {"USDJPY", -0.1m},   // Japan
            {"USDMXN", 4.25m},   // Mexico
            {"USDTRY", 7.5m},    // Turkey
            {"USDZAR", 7.0m}     // South Africa
        };

        // Subscribe to forex data for trading
        foreach (var ticker in _rates.Keys)
        {
            AddForex(ticker, Resolution.Daily);
        }
    }

    public override void OnData(Slice slice)
    {
        // Monthly rebalance checker
        if (_month == Time.Month) return;
        _month = Time.Month;

        // Long the pair with highest interest rate and sell the pair with the lowest to earn the max monetary
inflation difference between the two
        var sortedRates = (from kvp in _rates orderby kvp.Value ascending select kvp.Key).ToArray();
        SetHoldings(sortedRates[0], -0.5);
        SetHoldings(sortedRates[sortedRates.Length-1], 0.5);
    }
}
```

Framework Algorithm Example

The following example algorithm implements a FOREX carry trade. It buys the FOREX pair of the country with the lowest interest rate and sells the FOREX pair of the country with the highest interest rate.

```C#
public class ForexCarryTradeAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(25000);

        Symbol toSymbol(string ticker)
        {
            return QuantConnect.Symbol.Create(ticker, SecurityType.Forex, Market.Oanda);
        }

        // We will use hard-coded interest rates for each base currency
        var rates = new Dictionary<Symbol, decimal>()
        {
            {toSymbol("USDAUD"), 1.5m},    // Australia
            {toSymbol("USDCAD"), 0.5m},    // Canada
            {toSymbol("USDCNY"), 4.35m},   // China
            {toSymbol("USDEUR"), 0.0m},    // Euro Area
            {toSymbol("USDINR"), 6.5m},    // India
            {toSymbol("USDJPY"), -0.1m},   // Japan
            {toSymbol("USDMXN"), 4.25m},   // Mexico
            {toSymbol("USDTRY"), 7.5m},    // Turkey
            {toSymbol("USDZAR"), 7.0m}     // South Africa
        };

        SetUniverseSelection(new ManualUniverseSelectionModel(rates.Keys));
        // A custom alpha model to emit insight according to interest rate
        SetAlpha(new IntestRatesAlphaModel(rates));
        // Equal size to capitalize the monetary value of quote currency only based on interest rate difference
        // For dollar-neutral to save margin cost
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel(Expiry.EndOfMonth));
    }
}

public class IntestRatesAlphaModel : AlphaModel
{
    // Variable to control the rebalancing time
    private int _month = -1;
    private Dictionary<Symbol, decimal> _rates;

    public IntestRatesAlphaModel(Dictionary<Symbol, decimal> rates)
    {
        _rates = rates;
    }

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        // Monthly rebalance
        if (_month == algorithm.Time.Month)
        {
            return Enumerable.Empty<Insight>();
        }
        _month = algorithm.Time.Month;

        // Long the pair with highest interest rate and sell the pair with the lowest to earn the max monetary
inflation difference between the two
        var sortedRates = (from kvp in _rates orderby kvp.Value ascending select kvp.Key).ToArray();

        return Insight.Group(
            Insight.Price(sortedRates[0], Expiry.EndOfMonth, InsightDirection.Up),
            Insight.Price(sortedRates[sortedRates.Length-1], Expiry.EndOfMonth, InsightDirection.Down)
        );
    }
}
```

Data Point Attributes

The FOREX dataset provides `QuoteBar` and `Tick` objects.

QuoteBar Attributes

`QuoteBar` objects have the following attributes:

Tick Attributes

`Tick` objects have the following attributes:

QuantConnect

Fear and Greed

Introduction

The Fear and Greed dataset by QuantConnect provides data points to quantify the degree of fear and greed in the US Equity market. The data starts in 2014 and is delivered on a daily frequency, delayed by two days. This dataset is created by aggregating seven indicators, including market volatility, stock market breadth, and put-call ratios.

For more information about the Fear and Greed dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Getting Started

The following snippet demonstrates how to request data from the Fear and Greed dataset:

```C#
_datasetSymbol = AddData<FearGreedIndex>("FG").Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	July 2014
Data Density	Dense
Resolution	Daily
Timezone	New York

The latest daily data points are about two days since the VIX and SPX prices are inputs into the index and their latest daily prices are also two days old. New data is available at 9 AM EST.

Requesting Data

To add Fear and Greed data to your algorithm, call the `AddData<FearGreedIndex>` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```C#
public class FearAndGreedDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _datasetSymbol = AddData<FearGreedIndex>("FG").Symbol;
    }
}
```

Accessing Data

To get the current Fear and Greed data, index the current `Slice` with the dataset `Symbol` . `Slice` objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var dataPoint = slice[_datasetSymbol];
        Log($"{_datasetSymbol} value at {slice.Time}: {dataPoint.Value}");
    }
}
```

Historical Data

To get historical Fear and Greed data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
var history = History<FearGreedIndex>(_datasetSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscription

To remove a subscription to the Fear and Greed dataset, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_datasetSymbol);
```

Example Applications

The Fear and Greed dataset quantifies the degree of fear and greed in the US Equity market. Example use cases include the following strategies:

- Accumulate long-term positions when the market is fearful (the index is low).
- Short risky assets when the market is greedy.
- Trade butterfly Option strategies based on the market volatility indicator.

Classic Algorithm Example

The following example algorithm buys SPY when the Fear and Greed index is less than 25 (extreme fear) and sells when it's above 75 (extreme greed).

```C#
public class FearAndGreedExampleAlgorithm : QCAlgorithm
{
    private Equity _equity;
    private Symbol _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Add the market index to trade.
        _equity = AddEquity("SPY", Resolution.Daily);
        // Add the Fear and Greed dataset.
        _datasetSymbol = AddData<FearGreedIndex>("FG").Symbol;
    }

    public override void OnData(Slice data)
    {
        // Wait until the Fear and Greed dataset has new data
        if (!data.ContainsKey(_datasetSymbol))
        {
            return;
        }
        var value = data[_datasetSymbol].Value;
        // Buy when the market is fearful.
        if (!_equity.Invested && value < 25)
        {
            SetHoldings(_equity.Symbol, 1);
        }
        // Sell when the market is greedy.
        else if (_equity.Invested && value > 75)
        {
            Liquidate();
        }
    }
}
```

Framework Example

The following example algorithm buys SPY when the Fear and Greed index is less than 25 (extreme fear) and sells when it's above 75 (extreme greed).

```C#
public class FearAndGreedExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Trade SPY on a daily basis.
        UniverseSettings.Resolution = Resolution.Daily;
        var symbols = new[] { QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA) };
        AddUniverseSelection(new ManualUniverseSelectionModel(symbols));
        // Add a custom Alpha model that emits insights based on the Fear and Greed dataset.
        AddAlpha(new FearAndGreedAlphaModel(this));
        // Set the equal weighted PCM so the algorithm places trades.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }
}

// Define the custom Alpha model.
public class FearAndGreedAlphaModel : AlphaModel
{
    // Create a list to track the current universe.
    private List<Security> _securities = new List<Security>();
    private Symbol _datasetSymbol;
```

```
    public FearAndGreedAlphaModel(QCAlgorithm algorithm)
    {
        // Add the Fear and Greed dataset.
        _datasetSymbol = algorithm.AddData<FearGreedIndex>("FG").Symbol;
    }

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice data)
    {
        // Wait until the Fear and Greed dataset has new data.
        if (!data.ContainsKey(_datasetSymbol))
        {
            return Enumerable.Empty<Insight>();
        }
        // Wait until the Fear and Greed index is extreme.
        var value = data[_datasetSymbol].Value;
        if (value >= 25 && value <= 75)
        {
            return Enumerable.Empty<Insight>();
        }

        var insights = new List<Insight>();
        foreach (var security in _securities)
        {
            InsightDirection direction;
            // Buy when the market is fearful.
            if (!security.Invested && value < 25)
            {
                direction = InsightDirection.Up;
            }
            // Sell when the market is greedy.
            else if (security.Invested && value > 75)
            {
                direction = InsightDirection.Flat;
            }
            else
            {
                continue;
            }
            insights.Add(Insight.Price(security.Symbol, TimeSpan.FromDays(2 * 365), direction));
        }
        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        // When assets enter the universe, save them.
        foreach (var security in changes.AddedSecurities)
        {
            _securities.Add(security);
        }
        // When assets leave the universe, remove them.
        foreach (var security in changes.RemovedSecurities)
        {
            _securities.Remove(security);
        }
    }
}
```

Data Point Attributes

The Fear and Greed dataset provides `FearGreedIndex` objects, which have the following attributes:

QuantConnect

International Future Universe

Introduction

The International Future Universe dataset by QuantConnect lists the available International Future contracts, their daily trading volume, and Open Interest. The data covers 4 contracts (DAX, FESX, HSI, and NKD), starts in July 1998, and is delivered on daily frequency. This dataset is created by monitoring the trading activity on the EUREX, HKFE, and CME.

This dataset depends on the US Futures Security Master dataset because the US Futures Security Master dataset contains information on symbol changes of the contracts.

This dataset `does not` contain market data. For market data, see International Futures by TickData and US Futures by AlgoSeek for NKD.

For more information about the International Future Universe dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Getting Started

The International Futures Universe dataset provides data for contract filtering/selection:

```C#
var hsi= AddFuture(Futures.Indices.HangSeng, Resolution.Minute)           // "HSI";
hsi.SetFilter(0, 90);
var dax = AddFuture(Futures.Indices.Dax, Resolution.Minute)               // "DAX";
dax.SetFilter(0, 90);
var fesx = AddFuture(Futures.Indices.EuroStoxx50, Resolution.Minute)      // "FESX";
fesx.SetFilter(0, 90);
var nkd = AddFuture(Futures.Indices.Nikkei225Dollar, Resolution.Minute)   // "NKD";
nkd.SetFilter(0, 90);
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	July 1998 (for details, see `Supported Assets`)
Coverage	4 Contracts
Data Density	Dense
Resolution	Daily
Timezone	Various (for details, see `Supported Assets`)
Market Hours	Regular and Extended

Requesting Data

To add International Future Universe data to your algorithm, call the `AddFuture` method. Save a reference to the Future object so you can access the data later in your algorithm. To define which contracts should be in your universe, specify the filter when requesting the Future data.

The `AddFuture` method provides a daily stream of Future chain data. To get the most recent daily chain, call the `FuturesChain` method with the underlying Future Symbol. The `FuturesChain` method returns data on all the tradable contracts, not just the contracts that pass your universe filter.

```
C#
public class InternationalFuturesDataAlgorithm : QCAlgorithm
{
    private Future _future;

    public override void Initialize()
    {
        SetStartDate(2013, 12, 20);
        SetEndDate(2014, 2, 20);
        SetCash(1000000);
        UniverseSettings.Asynchronous = true;
        var future = AddFuture(Futures.Indices.HangSeng);
        // Set our contract filter for this Future chain.
        _future.SetFilter((universe) => universe.StandardsOnly().FrontMonth());
        // Get the entire Futures chain for the current day.
        var chain = FuturesChain(_future.Symbol);
    }
}
```

For more information about creating Future Universes, see Futures .

Accessing Data

For information about accessing International Future Universe data, see Futures .

Historical Data

You can get historical International Future Universe data in an algorithm and the Research Environment.

Historical Data In Algorithms

To get historical International Future Universe data in an algorithm, call the `History<FutureUniverse>` method with continuous Future `Symbol` and a lookback period. This method returns the all the available contracts for each trading day, not the subset of contracts that pass your universe filter. If there is no data for the period you requested, the history result is empty.

```C#
// Add the Future and save a reference to it.
var future = AddFuture(Futures.Indices.HangSeng);

// Get historical data.
var history = History<FutureUniverse>(future.Symbol, 5);
```

For more information about historical International Future Universe data in algorithms, see Contracts .

Historical Data In Research

To get historical International Future Universe data in the Research Environment, call the `History<FutureUniverse>` method with continuous Future `Symbol` and a time period. This method returns the all the available contracts for each trading day, not the subset of contracts that pass your universe filter. If there is no data for the period you requested, the history result is empty.

```C#
// Add the Future and save a reference to it.
var future = qb.AddFuture(Futures.Indices.HangSeng);

// Get historical data.
var history = qb.History<FutureUniverse>(future.Symbol, new DateTime(2025, 1, 1), new DateTime(2025, 4, 1));
```

For more information about historical International Future Universe data in the Research Environment, see Daily Prices History .

Supported Assets

The following table shows the available Futures:

Ticker	Future	Start Date	Time Zone	Currency
HSI	Hang Seng Index Futures	Jan 2010	Asia/Hong Kong	HKD
DAX	German Index Futures	Jan 2010	Europe/Berlin	EUR
FESX	EURO STOXX 50 Index Futures	Jul 1998	Europe/Berlin	EUR
NKD	Nikkei 225 Index Futures	Jan 2007	America/Chicago	USD

Example Applications

The International Futures Universe dataset enables you to design Futures strategies accurately. Examples include the following strategies:

- Buying the Futures contract with the most open interest to reduce slippage and market impact
- Trade speculation on an International Index
- Trading bull calendar spreads to reduce volatility and margin requirements

Classic Algorithm Example

The following example algorithm uses the ZigZag indicator to determine the trend of Hang Seng Index. It then trades the Index with HSI Futures.

```C#
public class InternationalFuturesDataAlgorithm : QCAlgorithm
{
    private Future _hsiFuture;
    private ZigZag _zz;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Set the time zone to HKT to make it more comparable with the exchange.
        SetTimeZone(TimeZones.HongKong);
        // Set the account currency as HKD to trade HSI Futures.
        SetAccountCurrency("HKD", 1000000);

        // Seed the last price of the contracts for filling.
        Settings.SeedInitialPrices = true;

        // Request HSI Futures to trade.
        // Note that we will trade the contract with the highest open interest for liquidity.
        _hsiFuture = AddFuture(
            Futures.Indices.HangSeng,
            extendedMarketHours: true,
            dataMappingMode: DataMappingMode.LastTradingDay,
            contractDepthOffset: 0
        );
        // Adds contracts that expiry within 90 days. We will trade the farthest contract
        _hsiFuture.SetFilter(0,90);
        // Request the corresponding underlying index for feeding indicators for trade signal generation.
        var hsiIndex = AddIndex("HSI").Symbol;

        // Create a ZigZag indicator to trade Hang Seng Index price pivot points.
        _zz = ZZ(hsiIndex, 0.15m, 5, Resolution.Daily);
        // Warm up indicator for immediate readiness to trade.
        WarmUpIndicator(hsiIndex, _zz, Resolution.Daily);
    }

    public override void OnData(Slice slice)
    {
        // Only place trade if the Future contracts is in market opening hours to avoid stale fills.
        if (IsMarketOpen(_hsiFuture.Symbol) && _zz.IsReady)
        {
            var pivot = _zz.PivotType;
            FuturesChain chain;
            // If the last pivot point is low, the current trend is increasing after this low point.
            if (pivot == PivotPointType.Low && slice.FutureChains.TryGetValue(_hsiFuture.Symbol, out chain))
            {
                var contract = chain.OrderBy(x => x.Expiry).LastOrDefault()?.Symbol;
                SetHoldings(contract, 0.2m);
            }
            // If the last pivot point is high, the current trend decreases after this high point.
            if (pivot == PivotPointType.High && slice.FutureChains.TryGetValue(_hsiFuture.Symbol, out chain))
            {
                var contract = chain.OrderBy(x => x.Expiry).LastOrDefault()?.Symbol;
                SetHoldings(contract, -0.2m);
            }
        }
    }
}
```

Data Point Attributes

The International Future Universe dataset provides `FutureFilterUniverse` and `FutureUniverse` objects.

FutureFilterUniverse Attributes

`FutureFilterUniverse` objects have the following attributes:

FutureUniverse Attributes

`FutureUniverse` objects have the following attributes:

QuantConnect

US ETF Constituents

Introduction

The US ETF Constituents dataset by QuantConnect tracks the constituents and weighting of US Equities in 2,650 ETF listings. The data starts in June 2009 and is delivered on a daily basis (monthly basis before January 2015). This dataset is created by tracking the host ETF websites and can be delayed by up to 1 week.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the US ETF Constituents dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 160,000 quants are served every month.

Getting Started

The following snippet demonstrates how to request data from the US ETF Constituents dataset:

```C#
public override void Initialize()
{
    UniverseSettings.Asynchronous = true;
    // Use the following method for a Classic Algorithm
    _universe = AddUniverse(Universe.ETF("SPY", Market.USA, UniverseSettings, ETFConstituentsFilter));

    var symbol = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
    // Use the following method for a Framework Algorithm
    AddUniverseSelection(new ETFConstituentsUniverseSelectionModel(symbol, UniverseSettings, ETFConstituentsFilter));
}
private IEnumerable<Symbol> ETFConstituentsFilter(IEnumerable <ETFConstituentUniverse> constituents)
{
    // Add all Symbols of the ETFConstituentUniverse
    return constituents.Select(x => x.Symbol);
}
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	June 2009
Asset Coverage	2,650 US ETF Listings
Data Density	Dense
Resolution	Daily (Monthly before Jan 2015)
Timezone	New York

Requesting Data

To add US ETF Constituents data to your algorithm, call the `AddUniverse` and `Universe.ETF` methods. To select which constituents occupy the universe, provide the ETF `Symbol` and a selection function.

```
C#
public class ETFConstituentUniverseAlgorithm : QCAlgorithm
{
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2018, 1, 1);
        SetEndDate(2020, 8, 25);
        SetCash(100000);
        UniverseSettings.Asynchronous = true;
        _universe = AddUniverse(Universe.ETF("SPY", UniverseSettings, ETFConstituentsFilter));
    }
}
```

For more information about universe settings, see Settings .

Accessing Data

To access the US ETF Constituent data, use the `ETFConstituentUniverse` objects in your selection function. The data is available in daily resolution. The `Symbol` objects you return from your selection function defines the universe constituents.

```
C#
public IEnumerable<Symbol> ETFConstituentsFilter(IEnumerable<ETFConstituentUniverse> constituents)
{
    foreach (var c in constituents)
    {
        Debug($"{c.EndTime} :: {c.LastUpdate} :: {c.Weight} :: {c.SharesHeld} :: {c.MarketValue}");
    }

    return constituents.Select(c => c.Symbol);
}
```

Historical Data

You can get historical universe data in an algorithm and in the Research Environment.

Historical Universe Data in Algorithms

To get historical universe data in an algorithm, call the `History` method with the `Universe` object and the lookback period. If there is no data in the period you request, the history result is empty.

```C#
var history = History(_universe, 30, Resolution.Daily);
foreach (var constituents in history)
{
    foreach (ETFConstituentUniverse constituent in constituents)
    {
        Log($"{constituent.Symbol} weight at {constituent.EndTime}: {constituent.Weight}");
    }
}
```

Historical Universe Data in Research

To get historical universe data in research, call the `UniverseHistory` method with the `Universe` object and the lookback period. The `UniverseHistory` returns the filtered universe. If there is no data in the period you request, the history result is empty.

```C#
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var constituents in universeHistory )
{
    foreach (ETFConstituentUniverse constituent in constituents)
    {
        Console.WriteLine($"{constituent.Symbol} weight at {constituent.EndTime}: {constituent.Weight}");
    }
}
```

You can call the `History` method in Research.

Supported ETFs

The following table shows the available ETFs:

ETF Tickers					
AADR	AAIT	AAWW	AAXJ	ABCS	ABEQ
ACCU	ACES	ACIM	ACIO	ACSG	ACSI
ACT	ACTV	ACWF	ACWI	ACWV	ACWX
ADFI	ADIV	ADME	ADRA	ADRD	ADRE
ADRU	AEMB	AESR	AFK	AFLG	AFMC
AFSM	AFTY	AGEM	AGG	AGGE	AGGH
AGGP	AGIH	AGII	AGLS	AGNG	AGOV
AGQ	AGRG	AGRH	AGT	AGZ	AHYB
AIA	AIEQ	AIIQ	AIQ	AIRR	ALFA
ALFI	ALTL	ALTS	ALTY	AMCA	AMER
AMID	AMLP	AMLX	AMOM	AMPS	AMZA
ANEW	ANGL	AOA	AOK	AOM	AOR
AOTG	APRZ	AQWA	ARCM	ARGT	ARKG

ARKK	ARKQ	ARKW	ARKX	ARVR	ASEA
ASET	ASHR	ASHX	AUGZ	AUSE	AUSF
AVDE	AVDV	AVEM	AVES	AVGE	AVIE
AVIV	AVLV	AVRE	AVSC	AVSD	AVSE
AVSU	AVUS	AVUV	AWAY	AXDI	AXEN
AXFN	AXHE	AXID	AXIT	AXJL	AXJS
AXJV	AXMT	AXSL	AXTE	AXUT	BAB
BABS	BATT	BBAG	BBAX	BBC	BBCA
BBCB	BBEU	BBH	BBHY	BBIN	BBLU
BBMC	BBP	BBRC	BBRE	BBSA	BBSC
BBUS	BCHP	BCNA	BDAT	BDH	BDRY
BECO	BEDZ	BEMO	BERN	BETZ	BFIT
BFOR	BFTR	BHH	BIB	BIBL	BICK
BIDS	BIGD	BIK	BIKR	BITQ	BITS
BIV	BIZD	BJK	BKCH	BKCI	BKEM
BKES	BKF	BKGI	BKIE	BKIS	BKLC
BKMC	BKSE	BKUS	BLCN	BLDG	BLES
BLHY	BLKC	BLLD	BLOK	BLV	BMED
BND	BNDC	BNDW	BNDX	BNE	BNGE
BOB	BOSS	BOTZ	BOUT	BPAY	BRF
BRGL	BRNY	BRZU	BSCB	BSCC	BSCD
BSCE	BSCF	BSCG	BSCH	BSCI	BSCJ
BSCK	BSEA	BSJC	BSJD	BSJE	BSJF
BSJG	BSJH	BSJI	BSJJ	BSV	BTAH
BTAL	BTEC	BTEK	BTHM	BUFD	BUFF
BUFG	BUFQ	BUFR	BUFT	BUG	BUL
BUYZ	BUZ	BUZZ	BWX	BWZ	BYLD
BYOB	BYTE	CACG	CALF	CARZ	CATH

CBOE	CBON	CCON	CCOR	CCXE	CDC
CDEI	CDL	CDX	CEFA	CEFS	CEMB
CEY	CEZ	CFA	CFCV	CFGE	CFO
CGDV	CGGO	CGGR	CGUS	CGW	CGXU
CHAU	CHB	CHEP	CHGX	CHIC	CHIH
CHII	CHIK	CHIL	CHIQ	CHIR	CHIS
CHIX	CHLD	CHNA	CIBR	CID	CIL
CIRC	CIU	CIZ	CLDL	CLIX	CLNR
CLOU	CLRG	CLSA	CLSC	CLSM	CLY
CLYH	CMBS	CMDT	CMF	CN	CNBS
CNCR	CNDA	CNDF	CNHX	CNRG	CNSF
CNTR	CNXT	COLX	COMB	COMG	COMT
CONG	COPX	COWZ	CPI	CQQQ	CRAK
CRBA	CRBI	CRBN	CRBQ	CRED	CRO
CROP	CRPT	CRUZ	CSA	CSB	CSD
CSF	CSJ	CSM	CSML	CTEC	CTEX
CTRU	CU	CUMB	CURE	CUT	CVIE
CVLC	CVMC	CVSE	CVY	CWEB	CWI
CWS	CXSE	CYA	CZA	DALI	DALT
DAPP	DAT	DAX	DBA	DBAP	DBAU
DBAW	DBB	DBBR	DBC	DBE	DBEF
DBEM	DBES	DBEU	DBEZ	DBGR	DBIF
DBIT	DBIZ	DBJP	DBKO	DBLV	DBMX
DBN	DBO	DBRE	DBSE	DBSP	DBU
DBUK	DDBI	DDEZ	DDIV	DDLS	DDM
DDWM	DEB	DECZ	DEEF	DEEP	DEF
DEFA	DEFN	DEHP	DEM	DEMG	DENT
DES	DESC	DEUS	DEW	DEWJ	DEZU

DFAC	DFAE	DFAI	DFAR	DFAS	DFAT
DFAU	DFAX	DFE	DFEM	DFEN	DFEV
DFGR	DFHY	DFIC	DFIS	DFIV	DFJ
DFLV	DFND	DFNL	DFNV	DFSE	DFSI
DFSU	DFSV	DFUS	DFUV	DGRE	DGRO
DGRS	DGRW	DGS	DGT	DHDG	DHS
DIA	DIEM	DIG	DIHP	DIM	DINT
DISV	DIV	DIVA	DIVB	DIVD	DIVI
DIVO	DIVS	DIVZ	DJD	DJIA	DKA
DLN	DLS	DMAT	DMDV	DMRE	DMRI
DMRL	DMRM	DMRS	DMXF	DND	DNH
DNL	DOG	DOL	DON	DOO	DOZR
DPK	DPST	DQML	DRIV	DRLL	DRN
DRSK	DRW	DSC	DSCF	DSG	DSI
DSPC	DSTL	DSTX	DSUM	DSV	DTD
DTEC	DTH	DTN	DTOX	DTRE	DUDE
DUHP	DURA	DUSA	DUSL	DVAL	DVEM
DVLU	DVOL	DVP	DVY	DVYA	DVYE
DWAQ	DWAS	DWAT	DWAW	DWCR	DWEQ
DWFI	DWIN	DWLD	DWLV	DWM	DWMC
DWPP	DWSH	DWTR	DWUS	DWX	DXD
DXGE	DXJ	DXJC	DXJF	DXJH	DXJR
DXJS	DXJT	DXKW	DXPS	DXUS	DYHG
DYLD	DYLS	DYNF	DZK	EAFD	EAOA
EAOK	EAOM	EAOR	EASG	EASI	EATZ
EBIZ	EBLU	EBND	ECH	ECLN	ECNS
ECON	ECOW	ECOZ	EDBI	EDC	EDEN
EDIV	EDOC	EDOG	EDOM	EDOW	EDUT

EEB	EEG	EEHB	EELV	EEM	EEMA
EEMD	EEME	EEML	EEMO	EEMS	EEMV
EEMX	EEN	EEO	EES	EEZ	EFA
EFAD	EFAV	EFAX	EFG	EFIV	EFN
EFNL	EFRA	EFV	EGIS	EGPT	EGRW
EGUS	EIDO	EINC	EIPX	EIRL	EIS
EKAR	EKG	EKH	ELG	ELR	ELV
EMAG	EMB	EMBB	EMBH	EMBU	EMCA
EMCD	EMCG	EMCH	EMCR	EMDD	EMDG
EMDI	EMDV	EMEM	EMER	EMEY	EMFM
EMFN	EMFQ	EMFT	EMG	EMGC	EMGD
EMGF	EMHD	EMHY	EMIF	EMLC	EMLP
EMM	EMMT	EMPW	EMQQ	EMRE	EMSG
EMSO	EMT	EMTL	EMV	EMXC	EMXF
EMZA	ENFR	ENGN	ENOR	ENRG	ENTR
ENY	ENZL	EOPS	EPHE	EPI	EPOL
EPP	EPRE	EPRF	EPRO	EPS	EPU
EQAL	EQIN	EQL	EQLT	EQRR	EQUL
EQWL	EQWM	EQWS	ERET	ERGF	ERM
ERSX	ERUS	ERX	ESG	ESGA	ESGD
ESGE	ESGF	ESGG	ESGL	ESGN	ESGS
ESGU	ESGV	ESGW	ESGY	ESIX	ESML
ESMV	ESNG	ESPO	ESR	ETHO	ETPA
EUDG	EUDV	EUFL	EUFN	EUMF	EUMV
EURL	EURZ	EUSA	EUSC	EUXL	EVAL
EVEN	EVX	EWA	EWAC	EWC	EWCO
EWD	EWEB	EWEF	EWEM	EWG	EWGS
EWH	EWHS	EWI	EWJ	EWK	EWL

EWM	EWMC	EWMD	EWN	EWO	EWP
EWQ	EWRE	EWRI	EWRM	EWRS	EWS
EWSC	EWSM	EWSS	EWT	EWU	EWUS
EWW	EWX	EWY	EWZ	EWZS	EXB
EXI	EXT	EYLD	EZA	EZJ	EZM
EZU	EZY	FAA	FAB	FAD	FAIL
FAN	FAS	FATT	FAUS	FBCG	FBCV
FBM	FBND	FBT	FBZ	FCA	FCAN
FCD	FCEF	FCG	FCHI	FCL	FCLD
FCOM	FCOR	FCPI	FCQ	FCTR	FCV
FCVT	FDD	FDEM	FDEV	FDG	FDHT
FDIG	FDIS	FDIV	FDL	FDLO	FDM
FDMO	FDN	FDNI	FDRR	FDRV	FDT
FDTS	FDV	FDVV	FDWM	FEBZ	FEDM
FEDX	FEEM	FEFN	FEG	FEHY	FEIG
FEM	FEMB	FEMS	FENY	FEP	FEUS
FEUZ	FEVR	FEX	FFHG	FFL	FFR
FFSG	FFTG	FFTI	FFTY	FGD	FGEM
FGM	FGRO	FHC	FHK	FHLC	FIDI
FIDU	FIL	FILL	FINU	FINX	FIO
FISR	FITE	FIVA	FIVG	FIW	FJP
FKL	FKO	FKU	FLAG	FLAU	FLAX
FLBR	FLCA	FLCH	FLCO	FLEE	FLEH
FLFR	FLG	FLGB	FLGR	FLHK	FLIN
FLIO	FLIY	FLJH	FLJP	FLKR	FLLA
FLLV	FLM	FLMB	FLMI	FLMX	FLN
FLOT	FLQD	FLQE	FLQG	FLQH	FLQL
FLQM	FLQS	FLRG	FLRT	FLRU	FLSA

FLSP	FLSW	FLTB	FLTR	FLTW	FLV
FLYT	FLZA	FM	FMAG	FMAT	FMB
FMET	FMF	FMIL	FMK	FMM	FMU
FMV	FNCF	FNCL	FNDA	FNDB	FNDC
FNDE	FNDF	FNDX	FNG	FNGG	FNI
FNIO	FNK	FNTC	FNX	FNY	FOC
FOMO	FONE	FOS	FOVL	FPA	FPE
FPFD	FPRO	FPX	FPXE	FPXI	FQAL
FRAK	FRDM	FREL	FRI	FRL	FRN
FRNW	FSBD	FSMD	FSMO	FSST	FSTA
FSYD	FSZ	FTA	FTAG	FTC	FTCS
FTEC	FTGC	FTGS	FTHI	FTLB	FTLS
FTQ	FTQI	FTRI	FTSL	FTSM	FTVA
FTW	FTXD	FTXG	FTXH	FTXL	FTXN
FTXO	FTXR	FTY	FUI	FUTY	FV
FVAL	FVC	FVD	FVI	FVL	FWDB
FWDD	FWDI	FXD	FXEU	FXG	FXH
FXI	FXL	FXN	FXO	FXR	FXU
FXZ	FYC	FYLG	FYT	FYX	FZB
GAA	GAF	GAL	GAMR	GARD	GASL
GASX	GBDV	GBF	GBGR	GBIL	GBLD
GBLO	GBUY	GCC	GCOR	GCOW	GDAT
GDIV	GDJJ	GDMA	GDNA	GDOC	GDVD
GDX	GDXJ	GDXX	GEM	GEMD	GENY
GERJ	GERM	GEUR	GEX	GFGF	GFIN
GFOF	GGEM	GHII	GHS	GHYB	GHYG
GIGB	GIGE	GII	GINN	GIVE	GIY
GK	GLCN	GLDE	GLIF	GLIN	GLOF

GLOV	GLRY	GMAN	GMET	GMF	GMFL
GMFS	GML	GMM	GMOM	GMTB	GNAT
GNMA	GNOM	GNR	GNRX	GOAT	GOAU
GOEX	GOP	GOVT	GPAL	GQRE	GREI
GREK	GRES	GRI	GRID	GRMY	GRNB
GRNR	GRPC	GRV	GSAX	GSD	GSEE
GSEU	GSEW	GSFP	GSG	GSGO	GSID
GSIE	GSIG	GSJY	GSLC	GSMA	GSPY
GSRA	GSSC	GSST	GSUS	GSY	GTAA
GTEK	GTIP	GTO	GULF	GUNR	GUR
GURU	GUSA	GUSH	GVAL	GVI	GVIP
GVT	GWL	GWX	GXC	GXF	GXG
GXTG	GYEN	GYLD	HACK	HACV	HACW
HAHA	HAIL	HAO	HAP	HAPI	HAPY
HART	HAUD	HAUZ	HAWX	HBTA	HBU
HCOM	HCRF	HDAW	HDEE	HDEF	HDEZ
HDG	HDGE	HDGI	HDIV	HDLS	HDMV
HDRO	HDUS	HDV	HDWM	HDWX	HECO
HEDJ	HEEM	HEET	HEFA	HEFV	HEGE
HEGJ	HELX	HEMV	HEQT	HERD	HERO
HEUS	HEUV	HEWC	HEWG	HEWI	HEWJ
HEWL	HEWP	HEWU	HEWW	HEWY	HEZU
HFEZ	HFGO	HFND	HFXE	HFXI	HFXJ
HGEM	HGEU	HGI	HGSD	HHH	HIBL
HIDE	HILO	HIPR	HIPS	HISF	HJEN
HJPX	HKK	HLAL	HLGE	HMTM	HNDL
HOLD	HOM	HOMZ	HOTL	HSCZ	HSMV
HSPX	HTEC	HTUS	HUSE	HUSV	HVAL

HVOL	HVPW	HYD	HYDD	HYDR	HYEM
HYG	HYGH	HYGI	HYGV	HYHG	HYIH
HYLD	HYLG	HYLS	HYLV	HYXU	IAH
IAI	IAK	IAT	IAU	IBB	IBBJ
IBBQ	IBCB	IBCC	IBCD	IBCE	IBD
IBDA	IBDB	IBDC	IBDD	IBDF	IBDH
IBDJ	IBDK	IBDL	IBDM	IBDN	IBDO
IBDP	IBDQ	IBDT	IBHA	IBHB	IBHC
IBHD	IBHE	IBLC	IBLN	IBMD	IBME
IBMF	IBMG	IBMH	IBMI	IBND	IBRN
IBUY	ICAN	ICAP	ICF	ICLN	ICOL
ICOW	ICSH	ICVT	IDAT	IDEV	IDHB
IDHD	IDHQ	IDLB	IDLV	IDMO	IDNA
IDOG	IDRV	IDU	IDV	IDVO	IDX
IECS	IEDI	IEF	IEFA	IEFN	IEHS
IEI	IEIH	IEIL	IEIS	IELG	IEME
IEMG	IEO	IESM	IETC	IEUR	IEUS
IEV	IEZ	IFAS	IFEU	IFGL	IFLY
IFNA	IFRA	IFSM	IFV	IGBH	IGE
IGEM	IGF	IGHG	IGIB	IGM	IGN
IGOV	IGRO	IGSB	IGU	IGV	IGW
IHAK	IHDG	IHE	IHF	IHI	IHY
IIH	IJH	IJJ	IJK	IJNK	IJR
IJS	IJT	ILDR	ILF	ILTB	IMFL
IMOM	IMTM	INCO	INDA	INDF	INDL
INDS	INDY	INFL	INFR	INKM	INNO
INTF	INXX	IOIL	IOO	IPAC	IPAY
IPD	IPF	IPFF	IPK	IPKW	IPN

IPO	IPOS	IPS	IPU	IPW	IQDE
IQDF	IQDG	IQDY	IQIN	IQLT	IQM
IQSI	IQSU	IRBO	IRO	IRV	IRY
ISCF	ISDS	ISDX	ISEM	ISHG	ISHP
ISI	ISMD	ISRA	IST	ISTB	ISVL
ISZE	ITA	ITAN	ITB	ITEQ	ITF
ITIP	ITM	ITOT	IUS	IUSB	IUSG
IUSS	IUSV	IVAL	IVDG	IVE	IVEG
IVES	IVLC	IVLU	IVOG	IVOL	IVOO
IVOV	IVRA	IVSG	IVV	IVW	IWB
IWC	IWD	IWF	IWFH	IWIN	IWL
IWM	IWN	IWO	IWP	IWR	IWS
IWTR	IWV	IWW	IWX	IWY	IWZ
IXC	IXG	IXJ	IXN	IXP	IXUS
IYC	IYE	IYF	IYG	IYH	IYJ
IYK	IYLD	IYM	IYR	IYT	IYW
IYY	IYZ	IZRL	JANZ	JAVA	JBBB
JCO	JCPB	JCPI	JCTR	JDG	JDIV
JEMA	JEPI	JETS	JGLD	JGRO	JHCB
JHCS	JHDV	JHEM	JHID	JHMA	JHMC
JHMD	JHME	JHMF	JHMH	JHMI	JHML
JHMM	JHMS	JHMT	JHMU	JHPI	JHSC
JIDA	JIG	JIRE	JKD	JKE	JKF
JKG	JKH	JKI	JKJ	JKK	JKL
JMBS	JMEE	JMIN	JMOM	JMST	JMUB
JNUG	JOET	JPED	JPEM	JPEU	JPGB
JPGE	JPHF	JPHY	JPIE	JPIN	JPMB
JPME	JPMV	JPNL	JPRE	JPSE	JPST

JPUS	JQUA	JRE	JRNY	JSCP	JSMD
JSML	JUSA	JUST	JVAL	JXI	JZRO
KALL	KARS	KBA	KBE	KBUY	KBWB
KBWC	KBWD	KBWI	KBWP	KBWR	KBWX
KBWY	KCE	KCNY	KDFI	KDIV	KEJI
KEMP	KEMQ	KEMX	KESG	KFVG	KFYP
KGHG	KGRN	KGRO	KIE	KLCD	KLD
KLDW	KLEM	KLIP	KLNE	KME	KMED
KMLM	KNG	KNGS	KNOW	KOCG	KOIN
KOKU	KOL	KOMP	KORP	KORU	KRBN
KRE	KRMA	KROO	KROP	KRU	KSCD
KURE	KVLE	KWEB	KWT	KXI	LABU
LALT	LATM	LBJ	LBTA	LCTD	LCTU
LDEM	LDRI	LDRS	LDSF	LEAD	LEGR
LEMB	LEND	LFEQ	LGEM	LGLV	LGOV
LIT	LIV	LKOR	LLDM	LLEM	LLSC
LLSP	LMBS	LNGR	LOPX	LOUP	LOWC
LQD	LQDH	LRGE	LRGF	LRNZ	LSAF
LSAT	LSLT	LTL	LUXE	LVHB	LVHD
LVHE	LVHI	LVIN	LVL	LVOL	LVUS
LWPE	LYFE	MAAX	MAGA	MAKX	MAMB
MARB	MARZ	MATF	MATH	MATL	MAUI
MBB	MBCC	MBOX	MBSD	MCEF	MCHI
MCRO	MCSE	MDCP	MDD	MDEV	MDIV
MDLL	MDY	MDYG	MDYV	MEAR	MEME
MES	META	METV	MEXX	MFDX	MFEM
MFMS	MFUL	MFUS	MGC	MGK	MGV
MID	MIDE	MIDF	MIDU	MIDZ	MILN

MINC	MISL	MJ	MJUS	MKH	MLN
MLPA	MLPX	MMIN	MMIT	MMLG	MMSC
MMTM	MNA	MNM	MOAT	MOHR	MOM
MONY	MOO	MOOD	MOON	MORT	MOTE
MOTG	MOTI	MOTO	MPRO	MRAD	MRGR
MSGR	MSOS	MTK	MTUM	MUB	MULT
MUSI	MVIN	MVP	MVPS	MVV	MXDU
MXI	MZG	MZN	MZO	NACP	NAIL
NANR	NASH	NASI	NDIV	NDJI	NDVG
NEAR	NEED	NERD	NETL	NFLT	NFO
NFRA	NIFE	NIWM	NLR	NOBL	NOMO
NORW	NSCS	NSPI	NSPY	NTKI	NTSE
NTSI	NUAG	NUBD	NUCL	NUDM	NUDV
NUEM	NUGO	NUGT	NUHY	NULC	NULG
NULV	NUMG	NUMV	NURE	NUSA	NUSC
NUSI	NVQ	NWLG	NXTE	NXTG	NY
NYC	NYCC	NYF	NZAC	OASI	OBOR
OCEN	OCIO	OEF	OEUR	OEW	OGEM
OGIG	OIH	OLD	OMFL	OMFS	OMOM
OND	ONEF	ONEK	ONEO	ONEQ	ONEV
ONEY	ONG	ONLN	ONOF	ONTL	OOTO
OPD	OQAL	ORG	OSCV	OSIZ	OTP
OTR	OUSA	OUSM	OVB	OVF	OVL
OVLC	OVLH	OVLU	OVM	OVOL	OVS
OVT	OWNS	OYLD	PABU	PACA	PAF
PAGG	PALC	PAMC	PAO	PAVE	PAWZ
PBD	PBDM	PBE	PBEE	PBJ	PBP
PBS	PBSM	PBTQ	PBUS	PBW	PCA

PCEF	PDBC	PDEV	PDN	PDP	PEGA
PEJ	PEK	PEX	PEXL	PEY	PEZ
PFA	PFF	PFFA	PFFD	PFFR	PFI
PFIG	PFIX	PFM	PFUT	PFXF	PGAL
PGF	PGHY	PGJ	PGRO	PGX	PHB
PHDG	PHO	PHYL	PIC	PICB	PICK
PID	PIE	PILL	PIN	PINK	PIO
PIQ	PIV	PIZ	PJB	PJF	PJG
PJM	PJP	PKB	PKN	PKOL	PKW
PLAT	PLCY	PLDR	PLND	PLRG	PLTL
PLTM	PLW	PMA	PMNA	PMOM	PMPT
PMR	PNQI	PNXQ	POTX	PPA	PPH
PPI	PPLC	PPSC	PQBW	PQDI	PQIN
PQLC	PQSC	PQSG	PQSV	PQY	PQZ
PRAY	PREF	PRF	PRFZ	PRME	PRN
PRNT	PSAU	PSC	PSCC	PSCD	PSCE
PSCF	PSCH	PSCI	PSCM	PSCT	PSCU
PSDN	PSET	PSFF	PSI	PSIL	PSJ
PSK	PSL	PSMB	PSMC	PSMG	PSMM
PSP	PSQ	PSR	PSTL	PSY	PTE
PTEU	PTF	PTH	PTIN	PTJ	PTLC
PTMC	PTNQ	PTO	PTRP	PUI	PUTW
PUW	PVAL	PVI	PWB	PWC	PWJ
PWND	PWO	PWP	PWT	PWV	PWY
PWZ	PXE	PXF	PXH	PXI	PXJ
PXLC	PXLG	PXLV	PXMC	PXMG	PXMV
PXN	PXQ	PXR	PXSC	PXSG	PXSV
PXUS	PY	PYH	PYZ	PZA	PZD

PZI	PZJ	PZT	QABA	QAI	QARP
QAT	QCAN	QCLN	QCLR	QDEF	QDEM
QDF	QDIV	QDPL	QDXU	QDYN	QED
QEFA	QEH	QEM	QEMM	QGBR	QGEM
QGRO	QGTA	QID	QINC	QINT	QLC
QLD	QLS	QLT	QLTA	QLTB	QLTC
QLV	QLVD	QLVE	QMJ	QMN	QMOM
QPT	QPX	QQC	QQD	QQEW	QQJG
QQMG	QQQ	QQQA	QQQE	QQQJ	QQQM
QQQN	QQQQ	QQXT	QRFT	QRMI	QSY
QTEC	QTR	QTUM	QUAL	QUS	QVAL
QVM	QVML	QVMM	QVMS	QWLD	QXGG
QXMI	QXRR	QXTR	QXUS	QYLD	QYLG
RAAX	RAFE	RALS	RAVI	RAYS	RBIN
RBL	RBLD	RBUS	RCD	RDIV	RDMX
RDOG	RDVI	RDVY	RECS	REDV	REEM
REET	REFA	REGL	REIT	REM	REMG
REMX	RESE	RESI	RESP	RETL	REVS
REZ	RFAP	RFDA	RFDI	RFEM	RFEU
RFF	RFFC	RFG	RFUN	RFV	RGI
RGLB	RGRO	RHS	RIDV	RIET	RIGS
RIGZ	RING	RISN	RKH	RLY	RNDM
RNDV	RNEM	RNLC	RNMC	RNRG	RNSC
ROAM	ROB	ROBO	ROBT	ROCI	RODE
RODM	ROGS	ROI	ROKT	ROM	ROMO
ROOF	RORE	RORO	ROSC	ROUS	RPG
RPQ	RPV	RPX	RRF	RRGR	RSCO
RSP	RSPE	RSPY	RSU	RSUN	RSX

RSXJ	RTH	RTL	RTM	RTR	RTYD
RULE	RUSL	RUSS	RVRS	RWCD	RWDC
RWDE	RWED	RWG	RWGV	RWIU	RWJ
RWK	RWL	RWLS	RWM	RWO	RWR
RWSL	RWUI	RWV	RWVG	RWW	RWX
RXI	RXL	RYE	RYF	RYH	RYJ
RYLD	RYLG	RYT	RYU	RZG	RZV
SAA	SAEF	SAGG	SATO	SAVN	SBEU
SBIO	SBUS	SCAP	SCHA	SCHB	SCHC
SCHD	SCHE	SCHF	SCHG	SCHH	SCHI
SCHJ	SCHK	SCHM	SCHO	SCHP	SCHQ
SCHR	SCHV	SCHX	SCHY	SCHZ	SCIF
SCIN	SCJ	SCLP	SCOG	SCTR	SCZ
SDAG	SDCI	SDEM	SDG	SDGA	SDIV
SDOG	SDOW	SDS	SDSI	SDVY	SDY
SEA	SECT	SEMI	SENT	SEPZ	SFY
SFYF	SFYX	SGDJ	SGDM	SGGG	SGQI
SH	SHBT	SHE	SHLD	SHMO	SHNY
SHOC	SHV	SHVY	SHY	SHYD	SHYG
SIL	SILJ	SIMS	SIPE	SIXA	SIXH
SIXL	SIXS	SIZ	SIZE	SKOR	SKYY
SLBT	SLDR	SLIM	SLOW	SLQD	SLT
SLVP	SLVY	SLX	SLY	SLYG	SLYV
SMB	SMCP	SMD	SMDV	SMDY	SMH
SMIG	SMIN	SMLE	SMLF	SMLL	SMLV
SMMD	SMMV	SMOG	SNPE	SNSR	SNUG
SOCL	SOLR	SOVB	SOXL	SOXQ	SOXX
SPAK	SPBC	SPCX	SPD	SPDV	SPDW

SPEM	SPEU	SPGM	SPGP	SPHB	SPHD
SPHQ	SPLG	SPLV	SPMD	SPMO	SPMV
SPQQ	SPRE	SPSB	SPSM	SPTM	SPUC
SPUN	SPUS	SPUU	SPVM	SPVU	SPXE
SPXH	SPXL	SPXN	SPXS	SPXT	SPXU
SPXV	SPY	SPYB	SPYC	SPYD	SPYG
SPYV	SPYX	SQEW	SQLV	SQQQ	SRET
SRVR	SSAM	SSFI	SSLY	SSO	SSPX
SSPY	SSUS	SSXU	STH	STIP	STLC
STLG	STLV	STMB	STOT	STRV	STSB
STXD	STXG	STXK	STXV	STXX	SUB
SUBZ	SULR	SUNY	SUPL	SURE	SUSA
SUSC	SUSL	SVAL	SVOL	SWAN	SWAR
SWH	SWIN	SXQG	SXUS	SYE	SYG
SYLD	SYTL	SYUS	SYV	SZNE	TAIL
TAN	TAO	TAWK	TAXF	TAXR	TBF
TBLU	TBT	TCHF	TCHI	TCHP	TCTL
TDD	TDH	TDIV	TDN	TDSA	TDSB
TDSC	TDSD	TDSE	TDTF	TDTT	TDV
TDVG	TDX	TECB	TECL	TEMP	TENG
TEQI	TERM	TETF	TFIV	TFLO	TFLT
TGEM	TGN	TGR	TGRW	THCX	THD
THHY	THNQ	THRK	TILT	TINT	TINY
TIP	TIPX	TLDH	TLEH	TLH	TLT
TLTD	TLTE	TMDV	TMF	TMFC	TMFE
TMFG	TMFM	TMFS	TMFX	TMW	TNA
TOK	TOKE	TOLZ	TONS	TOTL	TPAY
TPHD	TPHE	TPIF	TPLC	TPLE	TPOR

TPSC	TPYP	TQQQ	TRFM	TRND	TRPL
TRSK	TRTY	TSPA	TTAC	TTAI	TTFS
TTH	TUR	TUSA	TUTI	TUTT	TWEB
TWM	TWOK	TWON	TYD	TYLG	TYNE
TYNS	TZD	TZE	TZG	TZI	TZL
TZO	TZV	TZW	TZY	UAE	UAV
UBD	UBIO	UBOT	UBT	UCC	UCOM
UCYB	UDBI	UDIV	UDOW	UEVM	UGE
UGEM	UIVM	UJB	UK	UKF	UKK
UKW	ULQ	ULST	ULTR	ULVM	UMDD
UMI	UOP	UPRO	UPW	UPWD	URA
URE	URTH	URTY	USAI	USD	USEQ
USHG	USIG	USLB	USMC	USMF	USMV
USPX	USRT	USSD	USSG	UST	USVM
USWD	USXF	UTES	UTH	UTLF	UTLT
UTRN	UTSL	UUP	UVDV	UVG	UVT
UVU	UWC	UWM	UXI	UYG	UYM
VALQ	VALX	VAMO	VAW	VB	VBK
VBR	VCAR	VCIT	VCLN	VCLO	VCLT
VCR	VCSH	VDC	VDE	VEA	VEGA
VEGI	VERS	VESH	VETS	VEU	VFH
VFIN	VFLQ	VFMF	VFMO	VFMV	VFQY
VFVA	VGEM	VGFO	VGIT	VGK	VGSH
VGT	VHT	VICE	VIG	VIGI	VIOG
VIOO	VIOV	VIRS	VIS	VIXH	VLLV
VLML	VLSM	VLU	VLUE	VMOT	VNAM
VNLA	VNM	VNQ	VNQI	VO	VOE
VONE	VONG	VONV	VOO	VOOG	VOOV

VOT	VOX	VPC	VPL	VPN	VPOP
VPU	VR	VRAI	VRP	VSDA	VSGX
VSL	VSMV	VSPY	VSS	VT	VTC
VTHR	VTI	VTRN	VTV	VTWG	VTWO
VTWV	VUG	VV	VWID	VWO	VXF
VXUS	VYM	VYMI	WANT	WBAL	WBIA
WBIB	WBIC	WBID	WBIE	WBIF	WBIG
WBIH	WBII	WBIL	WBIN	WBIR	WBIT
WBIY	WCAT	WCBR	WCHN	WCLD	WDIV
WDNA	WDRW	WDTI	WEAR	WEBL	WEXP
WFH	WFVK	WGRO	WINN	WIP	WIZ
WKLY	WLDR	WLTH	WMCR	WMH	WNDY
WOMN	WOOD	WPS	WREI	WSKY	WTAI
WTRE	WTRX	WTV	WUGI	WUSA	WWJD
WWOW	WXSP	XAR	XB	XBB	XBI
XBUY	XCCC	XCEM	XCLR	XCOM	XDAT
XDNA	XES	XFIV	XGC	XHB	XHE
XHLF	XHMO	XHS	XHYC	XHYD	XHYE
XHYF	XHYH	XHYI	XHYT	XITK	XJH
XJR	XKFS	XKII	XKST	XLB	XLBS
XLBT	XLC	XLE	XLES	XLF	XLFS
XLG	XLI	XLIS	XLK	XLKS	XLP
XLPS	XLRE	XLSR	XLU	XLUS	XLV
XLVO	XLVS	XLY	XLYS	XME	XMHQ
XMLV	XMMO	XMPT	XMVM	XMX	XNTK
XONE	XOP	XOUT	XPH	XPND	XRLV
XRMI	XRO	XRT	XSD	XSHD	XSHQ
XSLV	XSMO	XSOE	XSVM	XSVN	XSW

XT	XTEN	XTH	XTL	XTN	XTR
XTRE	XTWO	XTWY	XUSA	XVOL	XVV
XWEB	XYLD	XYLG	YAO	YDIV	YESR
YINN	YLCO	YLD	YLDE	YMLI	YMLP
YOLO	YPRO	YPS	YUMY	YYY	ZCAN
ZDEU	ZGBR	ZGEN	ZHOK	ZIG	ZLRG
ZMLP	ZSML				

Example Applications

The ETF Constituents dataset provides an excellent source of tradable universes for strategies without selection bias. When you use an ETF universe, the original ETF can serve as an excellent benchmark for your strategy performance. Other use cases include the following:

- Creating an index-tracking algorithm for customized passive portfolio management
- Performing statistical arbitrage with the base ETF

Classic Algorithm Example

The following example algorithm creates a dynamic universe of the 10 largest US Equities in the SPY ETF. Each day, the algorithm forms a dollar-neutral and market-neutral portfolio by buying the 10 ETF constituents and shorting the SPY ETF.

```C#
public class ETFConstituentUniverseAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    // Create a dictionary to store the ETF weights each day.
    private Dictionary<Symbol, decimal> _weightBySymbol = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        // Add the SPY to trade.
        _spy = AddEquity("SPY").Symbol;
        // Add an ETF constituents universe that selects the large caps
        // in SPY. Save the universe object so you can get history
        // for the universe.
        var universe = AddUniverse(Universe.ETF(_spy, universeFilterFunc: SelectAssets));
        // Get historical universe data.
        var history = History(universe, 30, Resolution.Daily);
        // Show an example of iterating through the historical data.
        foreach (var constituents in history)
        {
            var t = constituents.EndTime;
            foreach (ETFConstituentUniverse constituent in constituents)
            {
                var symbol = constituent.Symbol;
                var weight = constituent.Weight;
            }
        }
        // Add a Scheduled Event to rebalance the portfolio each day.
        Schedule.On(DateRules.EveryDay(_spy), TimeRules.AfterMarketOpen(_spy, 1), Rebalance);
    }

    private IEnumerable<Symbol> SelectAssets(IEnumerable<ETFConstituentUniverse> constituents)
    {
        // Select the 10 largest stocks in the SPY.
        _weightBySymbol = constituents.OrderByDescending(c => c.Weight).Take(10)
            // Save the weights for position sizing.
            .ToDictionary(c => c.Symbol, c => c.Weight ?? 0m);
        // Return the selected assets.
        return _weightBySymbol.Keys;
    }

    private void Rebalance()
    {
        // Create a long-short portfolio to earn the excess return of the
        // top 10 weighted stocks from SPY.
        var spyWeight = _weightBySymbol.Values.Sum();
        var targets = new List<PortfolioTarget>() { new PortfolioTarget(_spy, -0.5m) };
        foreach (var kvp in _weightBySymbol)
        {
            targets.Add(new PortfolioTarget(kvp.Key, 0.5m * kvp.Value / spyWeight));
        }
        // Liquidate the stocks that aren't in top 10.
        SetHoldings(targets, true);
    }
}
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of the 10 largest US Equities in the SPY ETF. Each day, the algorithm forms a dollar-neutral and market-neutral portfolio by buying the 10 ETF constituents and shorting the SPY ETF.

```C#
public class ETFConstituentUniverseFrameworkAlgorithm : QCAlgorithm
{
    public Dictionary<Symbol, decimal> WeightBySymbol = [];

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        UniverseSettings.Asynchronous = false;
        UniverseSettings.Resolution = Resolution.Minute;
```

```
        UniverseSettings.Resolution = Resolution.Minute;

        // Add universe selection on SPY's constituents to select only from large cap stocks
        // Save the universe to access its members for historical data call
        var spy = AddEquity("SPY").Symbol;
        AddUniverseSelection(new ETFConstituentsUniverseSelectionModel(spy, UniverseSettings,
ETFConstituentsFilter));

        // Add alpha model that set normalized weight as investment insight
        AddAlpha(new ETFConstituentsAlphaModel(this, spy));

        // Set up portfolio construction model that invest by the insight weights
        SetPortfolioConstruction(new InsightWeightingPortfolioConstructionModel
        {
            // Avoid excessive rebalance on insight changes
            RebalanceOnInsightChanges = false
        });

        AddRiskManagement(new NullRiskManagementModel());

        SetExecution(new ImmediateExecutionModel());
    }

    private IEnumerable<Symbol> ETFConstituentsFilter(IEnumerable<ETFConstituentUniverse> constituents)
    {
        // The top 10 weighted securities are considered better active selections
        // Save the weights for position sizing
        WeightBySymbol = constituents.Where(c=> c.Weight.HasValue).OrderByDescending(c => c.Weight).Take(10)
            .ToDictionary(c => c.Symbol, c => c.Weight.Value);

        return WeightBySymbol.Keys;
    }
}

public class ETFConstituentsAlphaModel : AlphaModel
{
    private int _day = -1;
    private Symbol _etf;
    private ETFConstituentUniverseFrameworkAlgorithm _algorithm;

    public ETFConstituentsAlphaModel(ETFConstituentUniverseFrameworkAlgorithm algorithm, Symbol etf)
    {
        _etf = etf;
        _algorithm = algorithm;
    }

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        // Rebalance daily since selection is on daily basis
        if (_day == algorithm.Time.Day)
        {
            return [];
        }

        _day = algorithm.Time.Day;

        var insights = new List<Insight>();

        // Create a long-short portfolio to earn excess return of the top 10 weighted stocks from SPY
        var etfWeight = (double)_algorithm.WeightBySymbol.Values.Sum();
        if (etfWeight > 0)
        {
            // Invest half the portfolio by normalized weights of the top 10 constituents
            foreach(var kvp in _algorithm.WeightBySymbol)
            {
                insights.Add(Insight.Price(kvp.Key, Expiry.EndOfDay, InsightDirection.Up, weight:
(double)kvp.Value/etfWeight * 0.5));
            }

            // Short the other half with SPY, looking to profit from the active selection
            insights.Add(Insight.Price(_etf, Expiry.EndOfDay, InsightDirection.Down, weight: 0.5));
        }

        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        // Liquidate the ones not in top 10 weights
        var symbols = changes.RemovedSecurities.Where(x => x.Invested).Select(x => x.Symbol);
        algorithm.Liquidate(symbols, tag: "Removed From Universe");
    }
```

```
    }
}
```

Research Example

The following example lists ETF constituents with the greatest weight in the SPY:

```C#
var qb = new QuantBook();

// Add the ETF
var symbol = qb.AddEquity("SPY").Symbol;

// Add ETF Universe Selection
IEnumerable<Symbol> ETFConstituentsFilter(IEnumerable<ETFConstituentUniverse> constituents)
{
    // Take the top 10 weighted constituents
    return constituents
        .OrderByDescending(c => c.Weight)
        .Take(10)
        .Select(c => c.Symbol);
}

var universe = qb.AddUniverse(qb.Universe.ETF(spy, qb.UniverseSettings, ETFConstituentsFilter));

// Historical Universe data
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var constituents in universeHistory )
{
    foreach (ETFConstituentUniverse constituent in constituents)
    {
        Console.WriteLine($"{constituent.Symbol} weight at {constituent.EndTime}: {constituent.Weight}");
    }
}
```

Data Point Attributes

The ETF Constituents dataset provides `ETFConstituentUniverse` objects, which have the following attributes:

QuantConnect

US Equities Short Availability

Introduction

The US Equity Short Availability dataset provides the available shares for open short positions and their borrowing cost in the US Equity market. The data covers 10,500 US Equities, starts in January 2018, and is delivered on a daily frequency. This dataset is created using information from the exchanges.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the US Equities Short Availability dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Getting Started

The following snippets demonstrate how to request data from the US Equities Short Availability dataset.

Interactive Brokers Data

```C#
security.SetShortableProvider(new InteractiveBrokersShortableProvider());
```

Axos Clearing Data

```C#
security.SetShortableProvider(new LocalDiskShortableProvider("axos"));
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2018
Asset Coverage	10,500 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	New York

Requesting Data

To add US Equities Short Availability data to your algorithm, set the shortable provider of each US Equity in your algorithm.

```
C#

public class ShortAvailabilityDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        var security = AddEquity("AAPL");
        // Set shortable provider as IB
        security.SetShortableProvider(new InteractiveBrokersShortableProvider());
        _symbol = security.Symbol;
    }
}
```

Accessing Data

To check how many shares are available for a security to short, call the `ShortableQuantity` method of the `ShortableProvider`

```
C#

var shortableProvider = Securities[_symbol].ShortableProvider;
// Get the shortable quantity of the selected symbol at the selected time
var shortableQuantity = shortableProvider.ShortableQuantity(_symbol, Time);

// Check if there are a certain quantity of shares available
var quantity = 100;
var isShortableQuantity = Shortable(_symbol, quantity);
```

To check borrowing cost, call the `FeeRate` or `RebateRate` method of the `ShortableProvider`

```
C#

var feeRate = shortableProvider.FeeRate(_symbol, Time);
var rebateRate = shortableProvider.RebateRate(_symbol, Time);
```

To get valid borrowing rates, use the InteractiveBrokersShortableProvider .

Example Applications

The US Equities Short Availability dataset enables you to accurately design strategies harnessing information about short availability. Examples include the following use cases:

- Avoiding short orders that the brokerage will reject
- Selecting securities based on how many shares are available to short

Classic Algorithm Example

The following example algorithm shorts GameStop every day there are shares available to short. If the algorithm receives a margin call, it liquidates the position and start again on the next day.

C#

```
public class ShortAvailabilityDataAlgorithm : QCAlgorithm
{
    private Equity _equity;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000);
        // Seed the security price as the last known price, such that the price data is immediately available at
initial rebalance
        Settings.SeedInitialPrices = true;
        AddSecurityInitializer(security =>
        {
            // Set the shortable provider as your broker for accurate short reality modeling
            security.SetShortableProvider(new InteractiveBrokersShortableProvider());
        });

        _equity = AddEquity("GME");

        // Set up daily rebalance scheduled event, since shortable quantity is updated daily
        Schedule.On(
            DateRules.EveryDay(_equity.Symbol),
            TimeRules.AfterMarketOpen(_equity.Symbol, 10),
            Rebalance);
    }

    public void Rebalance()
    {
        var symbol = _equity.Symbol;

        // You can obtain the shortable quantity to decide the submission of a short order
        // Fee and rebate rate is also available, such that you can calculate the expected return and decide if the
margin is worthwhile
        Plot("Total Shortable Quantity", symbol, _equity.ShortableProvider.ShortableQuantity(symbol, Time) ?? 0m);
        Plot("Borrowing Cost", "Fee Rate", _equity.ShortableProvider.FeeRate(symbol, Time));
        Plot("Borrowing Cost", "Rebate Rate", _equity.ShortableProvider.RebateRate(symbol, Time));

        // Test whether we can short the desired quantity
        var quantity = CalculateOrderQuantity(symbol, -1m);
        if (quantity != 0 && Shortable(symbol, quantity))
        {
            MarketOrder(symbol, quantity);
        }
    }

    public override void OnMarginCallWarning()
    {
        Liquidate();
    }
}
```

Framework Algorithm Example

The following example algorithm shorts GameStop every day there are shares available to short. If the algorithm receives a margin call, it liquidates the position and start again on the next day.

C#

```
public class ShortAvailabilityDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000);
        // Seed the security price as the last known price, such that the price data is immediately available at
initial rebalance
        Settings.SeedInitialPrices = true;
        AddSecurityInitializer(security =>
        {
            // Set the shortable provider as your broker for accurate short reality modeling
            security.SetShortableProvider(new InteractiveBrokersShortableProvider());
        });

        SetUniverseSelection(new ManualUniverseSelectionModel(
            QuantConnect.Symbol.Create("GME", SecurityType.Equity, Market.USA)));

        // Emit down-direction insights to short all securities in the universe
        SetAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Down, TimeSpan.FromDays(1)));
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
        SetExecution(new ShortableExecutionModel());
    }

    // On Margin Call, emit flat-direction insights to liquidate the positions
    public override void OnMarginCallWarning()
    {
        EmitInsights(Securities
            .Where(kvp => kvp.Value.Invested)
            .Select(kvp => Insight.Price(kvp.Key, TimeSpan.FromDays(1), InsightDirection.Flat))
            .ToArray());
    }

    public class ShortableExecutionModel : ImmediateExecutionModel
    {
        private readonly PortfolioTargetCollection _targetsCollection = new PortfolioTargetCollection();

        public override void Execute(QCAlgorithm algorithm, IPortfolioTarget[] targets)
        {
            _targetsCollection.AddRange(targets);
            // for performance we check count value, OrderByMarginImpact and ClearFulfilled are expensive to call
            if (_targetsCollection.Count > 0)
            {
                foreach (var target in _targetsCollection.OrderByMarginImpact(algorithm))
                {
                    // calculate remaining quantity to be ordered
                    var quantity = OrderSizing.GetUnorderedQuantity(algorithm, target);
                    // If the quantity is negative, ensure that the security is shortable
                    if (quantity > 0 || algorithm.Shortable(target.Symbol, quantity))
                    {
                        algorithm.MarketOrder(target.Symbol, quantity);
                    }
                }

                _targetsCollection.ClearFulfilled(algorithm);
            }
        }
    }
}
```

Data Point Attributes

The US Equities Short Availability data is a `Symbol` /decimal pair for `ShortQuantity` , `FeeRate` , and `RebateRate` .

QuantConnect

US Equity Coarse Universe

Introduction

The US Equity Coarse Universe dataset by QuantConnect is a daily universe of all trading stocks in the US for a given day with the end of day price and volume. The data covers 30,000 US Equities in total, with approximately 8,000 Equities per day. The data starts in January 1998 and is delivered each trading day. This dataset is created by taking the closing auction price tick from the daily L1 trade and quote exchange dumps.

This dataset depends on the US Equities by AlgoSeek dataset because universe selection adds and removes market data subscriptions and on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

QuantConnect/LEAN combines the data of this dataset with MorningStar Fundamental data in runtime. You can use this dataset in local development without the Morningstar counterpart.

For more information about the US Equity Coarse Universe dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Getting Started

The following snippet demonstrates how to request data from the US Equity Coarse Universe dataset:

- Direct Access using the Security object

```C#
var equity = AddEquity("IBM");
var ibmFundamental = equity.Fundamentals;
```

- Direct Access using the Symbol object

```C#
var ibm = QuantConnect.Symbol.Create("IBM", SecurityType.Equity, Market.USA);
var ibmFundamental = Fundamentals(ibm);
```

- Universe Selection

```C#
public override void Initialize()
{
    // Universe selection
    _universe = AddUniverseSelection(new FundamentalUniverseSelectionModel(FundamentalFilterFunction));
}

public override List<Symbol> FundamentalFilterFunction(List<Fundamental> fundamental)
{
    // Sort all equities with price above $10 by dollar volume, take the top 10
    return (from f in fundamental
            where f.Price > 10
            orderby f.DollarVolume descending
            select f.Symbol).Take(10);
}
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 1998
Asset Coverage	30,000 US Equities
Data Density	Dense
Resolution	Daily
Timezone	New York

This dataset doesn't include Over-the-Counter (OTC) stocks.

Requesting Data

You don't need any special code to request US Coarse Fundamental Data. You can access the current and historical fundamental data for any of the US Equities that this dataset includes.

```C#
public class CoarseFundamentalDataAlgorithm : QCAlgorithm
{
    private Universe _universe;
    public override void Initialize()
    {
        SetStartDate(2021, 1, 1);
        SetEndDate(2021, 7, 1);
        SetCash(100000);
        UniverseSettings.Asynchronous = true;

        // Option 1: Subscribe to individual US Equity assets
        AddEquity("IBM");

        // Option 2A: Create a fundamental universe (classic version)
        _universe = AddUniverse(FundamentalFilterFunction);

        // Option 2B: Create a fundamental universe (framework version)
        AddUniverseSelection(new FundamentalUniverseSelectionModel(FundamentalFilterFunction));
    }
}
```

For more information about universe settings, see Settings .

Accessing Data

If you add a fundamental universe to your algorithm, you can access fundamental data in the universe selection function.

```
C#
public IEnumerable<Symbol> FundamentalFunction(IEnumerable<Fundamental> fundamental)
{
    var sortedByDollarVolume = fundamental
        .OrderByDescending(x => x.DollarVolume)
        .Take(3).ToList();

    foreach (var cf in sortedByDollarVolume)
    {
        Debug($"{cf.EndTime} :: {cf.Symbol} : {cf.AdjustedPrice} :: {cf.DollarVolume}");
    }

    return sortedByDollarVolume.Select(x => x.Symbol);
}
```

To get fundamental data for Equities in your algorithm, use the `Fundamentals` property of the `Equity` objects. The fundamental data represent the corporate fundamentals for the current algorithm time.

```
C#
var fundamentals = Securities[symbol].Fundamentals;
```

To get fundamental data for Equities, regardless of whether or not you have subscribed to them in your algorithm, call the `Fundamentals` method. If you pass one `Symbol` , the method returns a `Fundamental` object. If you pass a list of `Symbol` objects, the method returns a list of `Fundamental` objects.

```
C#
// Single asset
var ibm = QuantConnect.Symbol.Create("IBM", SecurityType.Equity, Market.USA);
var ibmFundamental = Fundamentals(ibm);

// Multiple assets
var nb = QuantConnect.Symbol.Create("NB", SecurityType.Equity, Market.USA);
var fundamentals = Fundamentals(new List<Symbol>{ nb, ibm }).ToList();
```

Historical Data

You can get historical fundamental data in an algorithm and in the Research Environment.

Historical Data in Algorithms

To get historical fundamental data in an algorithm, call the `History` method with `Fundamental` type and the Equity `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
var ibm = QuantConnect.Symbol.Create("IBM", SecurityType.Equity, Market.USA);

// Fundamental objects
var fundamentalHistory = History<Fundamental>(ibm, TimeSpan.FromDays(30));

// Fundamentals objects for all US Equities (including delisted companies)
var fundamentalsHistory = History<Fundamentals>(TimeSpan.FromDays(30));

// Collection of Fundamental objects for all US Equities (including delisted companies)
var collectionHistory = History(_universe, 30, Resolution.Daily);
foreach (var fundamental in collectionHistory)
{
    // Cast to Fundamental is required
    var mostLiquid = fundamental.OfType<Fundamental>().OrderByDescending(x => x.DollarVolume).Take(5);
}
```

Historical Universe Data in Research

To get historical universe data in the Research Environment, call the `UniverseHistory` method with the `Universe` object and the lookback period. This method returns the filtered universe. If there is no data in the period you request, the history result is empty.

```C#
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var fundamentals in universeHistory)
{
    foreach (Fundamental fundamental in fundamentals)
    {
        Console.WriteLine($"{fundamental.Symbol} dollar volume at {fundamental.EndTime}:
{fundamental.DollarVolume}");
    }
}
```

For more information about historical US Equity fundamental data, see Equity Fundamental Data .

Example Applications

The US Equity Coarse Universe dataset enables you to accurately design a universe of US Equities. Examples include the following strategies:

- Selecting securities with the largest dollar volume
- Selecting securities within a specific price range
- Selecting securities that have fundamental data available (together with MorningStar Fundamental data in `Fundamental` objects)

Classic Algorithm Example

The following example algorithm creates a dynamic universe of the three most liquid US Equities. Each time the universe changes, the algorithm forms an equal-weighted portfolio of the three companies.

C#

```
public class USEquityCoarseUniverseConstituentsDataAlgorithm : QCAlgorithm
{
    private int _numberOfSymbols = 3;
    private SecurityChanges _changes = SecurityChanges.None;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Set Asynchronous to true to improve speed performance
        UniverseSettings.Asynchronous = true;
        // Requesting data
        var universe = AddUniverse(SelectAssets);
        // Get historical data for the universe.
        var history = History(universe, 30, Resolution.Daily);
        foreach (var fundamentals in history)
        {
            // Iterate through each asset in the universe on this day
            // and access its data point attributes.
            foreach (Fundamental f in fundamentals)
            {
                var symbol = f.Symbol;
                var t = f.EndTime;
                var dollarVolume = f.DollarVolume;
            }
        }
    }

    public IEnumerable<Symbol> SelectAssets(IEnumerable<Fundamental> fundamental)
    {
        // Select the top traded stocks, since they are the most popular
        // with high capital flow.
        return fundamental.OrderByDescending(x => x.DollarVolume)
            .Take(_numberOfSymbols).Select(x => x.Symbol);
    }

    public override void OnData(Slice slice)
    {
        // If we have no changes, do nothing.
        if (slice.Bars.Count == 0 || _changes == SecurityChanges.None) return;
        // Liquidate removed securities, since they are not the most
        // popular stocks.
        foreach (var security in _changes.RemovedSecurities)
        {
            if (security.Invested)
            {
                Liquidate(security.Symbol);
            }
        }
        // We want 1/N allocation for each security in our universe to
        // evenly dissipate risk.
        foreach (var security in _changes.AddedSecurities)
        {
            SetHoldings(security.Symbol, 1m / _numberOfSymbols);
        }
        _changes = SecurityChanges.None;
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        _changes = changes;
    }
}
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of the three most liquid US Equities and forms an equal-weighted portfolio with them.

```C#
public class USEquityCoarseUniverseConstituentsDataAlgorithm : QCAlgorithm
{
    private int _numberOfSymbols = 3;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Set Asynchronous to true to improve speed performance
        UniverseSettings.Asynchronous = true;

        // Requesting data
        SetUniverseSelection(
            new FundamentalUniverseSelectionModel(FundamentalSelectionFunction));

        // we want 1/N allocation in each security in our universe to evenly dissipate risk
        SetAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(1)));
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());

        AddRiskManagement(new NullRiskManagementModel());

        SetExecution(new ImmediateExecutionModel());
    }

    public IEnumerable<Symbol> FundamentalSelectionFunction(IEnumerable<Fundamental> fundamental)
    {
        // Select the top traded stocks, since they are the most popular with high capital flow
        return fundamental.OrderByDescending(x => x.DollarVolume)
            .Take(_numberOfSymbols).Select(x => x.Symbol);
    }
}
```

Research Example

The following example lists the three most liquid US Equities:

```C#
var qb = new QuantBook();

// Add Fundamental Universe Selection
IEnumerable<Symbol> FundamentalSelectionFunction(IEnumerable<Fundamental> fundamentals)
{
    return fundamentals.OrderByDescending(x => c.DollarVolume).Take(10).Select(x => c.Symbol);
}

var universe = qb.AddUniverse(FundamentalSelectionFunction);

// Historical Universe data
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var fundamentals in universeHistory)
{
    foreach (Fundamental fundamental in fundamentals)
    {
        Console.WriteLine($"{fundamental.Symbol} dollar volume at {fundamental.EndTime}:
{fundamental.DollarVolume}");
    }
}
```

Data Point Attributes

The US Equity Coarse Universe dataset provides its data within `Fundamental` objects, which have the following . The attributes from `MarketCap` to `AssetClassification` listed in the following widget are NaN until this dataset is combined with the MorningStar Fundamental dataset.

QuantConnect

US Equity Option Universe

Introduction

The US Equity Option Universe dataset by QuantConnect lists the available US Equity Options contracts and the current Implied Volatility and Greeks. The data covers 4,000 Symbols, starts in January 2012, and is delivered on a daily update frequency. To create this dataset, we use our implementation of the forward tree pricing model, which accounts for the interest rate, dividend payments, and daily closing prices. The values in this dataset are the same values you can get from daily indicators with mirror Options.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes of the underlying security.

This dataset `does not` contain market data. For market data, see US Equity Options by AlgoSeek .

For more information about the US Equity Option Universe dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Getting Started

The following snippet demonstrates how to request data from the US Equity Options Universe dataset:

```C#
var option = AddOption("GOOG");
_optionSymbol = option.Symbol;
option.SetFilter(universe => universe.delta(0.4m, 0.6m));
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2012
Asset Coverage	4,000 Symbols
Data Density	Dense
Resolution	Daily
Timezone	New York

Requesting Data

To add US Equity Options Universe data to your algorithm, call the `AddOption` method. Save a reference to the Equity Option `Symbol` so you can access the data later in your algorithm. To define which contracts should be in your universe, call the `SetFilter` method of the Option object.

The `AddOption` method provides a daily stream of Option chain data. To get the most recent daily chain, call the `OptionChain` method with the underlying Equity Symbol. The `OptionChain` method returns data on all the tradable contracts, not just the contracts that pass your universe filter.

```C#
public class USEquityOptionsDataAlgorithm : QCAlgorithm
{
    private Symbol _optionSymbol;

    public override void Initialize()
    {
        SetStartDate(2020, 6, 1);
        SetEndDate(2021, 6, 1);
        SetCash(100000);
        UniverseSettings.Asynchronous = true;
        // Requesting data
        var option = AddOption("GOOG");
        _optionSymbol = option.Symbol;
        // Set our strike/expiry filter for this option chain
        option.SetFilter(OptionFilter);
        // Get the entire Option chain for the current day.
        var chain = OptionChain(option.Symbol.Underlying);
    }

    private OptionFilterUniverse OptionFilter(OptionFilterUniverse universe)
    {
        // Contracts can be filtered by greeks, implied volatility, open interest:
        return universe
            .Delta(0.5m, 1.5m)
            .Gamma(0.0001m, 0.0006m)
            .Vega(0.01m, 1.5m)
            .Theta(-2.0m, -0.5m)
            .Rho(0.5m, 3.0m)
            .ImpliedVolatility(1.0m, 3.0m)
            .OpenInterest(100, 500);
    }
}
```

The Equity resolution must be less than or equal to the Equity Option resolution. For example, if you set the Equity resolution to minute, then you must set the Equity Option resolution to minute, hour, or daily.

For more information about creating US Equity Option Universes, see Equity Options .

Accessing Data

For information about accessing US Equity Options Universe data, see Equity Options .

Historical Data

You can get historical US Equity Options Universe data in an algorithm and the Research Environment.

Historical Data In Algorithms

To get historical US Equity Options Universe data in an algorithm, call the `History<OptionUniverse>` method with the canonical Equity Option `Symbol` . This method returns data on all of the tradable contracts, not just the contracts that pass your universe filter. If there is no data in the period you request, the history result is empty.

```C#
// OptionUniverse objects
var history = History<OptionUniverse>(_optionSymbol, TimeSpan.FromDays(3)).ToList();
```

For more information about historical Equity Options Universe data in algorithms, see Historical Data .

Historical Data In Research

To get historical US Equity Options Universe data in the Research Environment, call the `History<OptionUniverse>` method with the canonical Option `Symbol` . This method returns data on all of the tradable contracts, not just the contracts that pass your universe filter.

```
C#
var qb = new QuantBook();
var option = qb.AddOption("GOOG");
var history = qb.History<OptionUniverse>(option.Symbol, new DateTime(2020, 6, 1), new DateTime(2020, 6, 6));
foreach (var chain in history)
{
    var endTime = chain.EndTime;
    var filteredContracts = chain.Data
        .Select(contract => contract as OptionUniverse)
        .Where(contract => contract.Greeks.Delta > 0.3m);
    foreach (var contract in filteredContracts)
    {
        var price = contract.Price;
        var iv = contract.ImpliedVolatility;
    }
}
```

For more information about historical Equity Options Universe data in the Research Environment, see Universes .

Supported Assets

To view the supported assets in the US Equity Options Universe dataset, see the Data Explorer .

Example Applications

The US Equity Options Universe dataset enables you to accurately design Option strategies. Examples include the following strategies:

- Buying put Options to hedge against downward price movement in positive Equity positions
- Exploiting arbitrage opportunities that arise when the price of Option contracts deviate from their theoretical value

Classic Algorithm Example

The following example algorithm subscribes to Google put options that fall within delta range between -1 and -0.95, open interest range between 10 and 1000, and expire within seven days. Within this Option chain, the algorithm holds the put Option contract that has the minimum delta (closest to -1) during market hour to hedge the underlying intra-day movement completely. It avoid volatility from sentiment and only earns from inter-day movement from longer-term factors. When the contract expires, the algorithm rolls over to the next contract that meets this criteria.

```
C#
public class USEquityOptionsUniverseAlgorithm : QCAlgorithm
{
    private Symbol _equity, _optionSymbol, _hedge;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(10000000);
        // Asynchronous can use computational resources efficiently
        UniverseSettings.Asynchronous = true;

        // Subscribe to the underlying for the underlying position
        // Set the data normalization mode to raw for strike price comparability
        _equity = AddEquity("GOOG", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;
        // Requesting option data and filter for the hedge candidates
        var option = AddOption(_equity);
```

```
        var option = AddOption(_equity);
        _optionSymbol = option.Symbol;
        option.SetFilter(OptionFilter);

        // Set scheduled event to buy a hedge option contract at market open to eliminate the intra-day movement
        Schedule.On(
            DateRules.EveryDay(_equity),
            TimeRules.AfterMarketOpen(_equity, 1),
            BuyHedgeContract
        );

        // Set a scheduled event to sell the hedge contract before market close, since we want to earn from inter-day
movement
        // Leave 2 minutes contingency to fill
        Schedule.On(
            DateRules.EveryDay(_equity),
            TimeRules.BeforeMarketClose(_equity, 2),
            SellHedgeContract
        );
    }

    private OptionFilterUniverse OptionFilter(OptionFilterUniverse universe)
    {
        // Select the contracts with delta very close to -1 and high open interest
        // This can effectively hedge most of the price change of the underlying and ensure the liquidity
        // Make sure the contract is expiring close for its tradbility
        return universe
            .IncludeWeeklys()
            .PutsOnly()
            .Expiration(2, 7)
            .Delta(-1m, -0.95m)
            .OpenInterest(10, 1000);
    }

    private void BuyHedgeContract()
    {
        if (CurrentSlice.OptionChains.TryGetValue(_optionSymbol, out var chain))
        {
            // Order the underlying if not hold, the order size should match the option contract
            // Order only if option chain data ready for hedging
            if (!Portfolio[_equity].Invested)
            {
                MarketOrder(_equity, Securities[_optionSymbol].SymbolProperties.ContractMultiplier);
            }

            // Get the contract with delta closest to -1 (lowest possible delta)
            var contract = chain.MinBy(x => x.Greeks.Delta);
            _hedge = contract.Symbol;
            // Buy 1 deep ITM put with delta close to -1 to eliminate the intraday movement
            MarketOrder(_hedge, 1);
        }
    }

    private void SellHedgeContract()
    {
        // Check if any hedge contract position, if so, liquidate before market close to expose to underlying
overnight movement
        if (_hedge != null)
        {
            Liquidate(_hedge);
            _hedge = null;
        }
    }
}
```

Framework Algorithm Example

The following example algorithm demonstrating a Gamma Scalping strategy through framework algorithm. It filters NVDA options with expiration between 30 to 90 days and open interest between 50 and 1000, due to liquidity concern and lower Gamma fluctuation. Assuming a significant upward trend is expecting, it orders a long straddle strategy from the strike price with Delta-neutral and the highest Gamma to earn the highest profit from upward underlying movement.

C#

```
public class USEquityOptionsUniverseFrameworkAlgorithm : QCAlgorithm
{
    public override void Initialize()
```

```
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        // Asynchronous can use computational resources efficiently
        UniverseSettings.Asynchronous = true;

        // Universe selection that select based on liquidity of the option contracts (by open interest)
        AddUniverseSelection(new EquityOptionsUniverseSelectionModel());
        // Custom alpha model that use Delta and Gamma to signal insights
        AddAlpha(new OptionDeltaGammaAlphaModel());
        // To maintain long-short size equal, use a PCM that order a single contract
        SetPortfolioConstruction(new SingleSharePortfolioConstructionModel());
    }
}

class EquityOptionsUniverseSelectionModel : OptionUniverseSelectionModel
{
    // Daily update with the SelectOptionChainSymbols function
    public EquityOptionsUniverseSelectionModel()
            : base(TimeSpan.FromDays(1), SelectOptionChainSymbols) {}

    private static IEnumerable<Symbol> SelectOptionChainSymbols(DateTime utcTime)
    {
        // Select only NVDA options as our focus, which is a highly traded volatile equity with great upward momentum
        return new[] {QuantConnect.Symbol.Create("NVDA", SecurityType.Option, Market.USA)};
    }

    protected override OptionFilterUniverse Filter(OptionFilterUniverse filter)
    {
        // To ensure the liquidity and tradability, make sure the option is not 0-DTE and have a fair open interest
        // A longer TTM will have lower Gamma thus more stable in the local delta-neutral position
        return filter
            .IncludeWeeklys()
            .Expiration(30, 90)
            .OpenInterest(50, 1000);
    }
}

class OptionDeltaGammaAlphaModel : AlphaModel
{
    // A day count variable to control the alpha model only trade once a day
    private int _day = -1;

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = new List<Insight>();

        if (_day != slice.Time.Day)
        {
            // We open position for each option underlying as separate bet
            foreach (var kvp in slice.OptionChains)
            {
                var chain = kvp.Value;

                // For theta-neutral, select the same expiry for both call and put
                var expiry = chain.Min(x => x.Expiry);
                var contracts = chain.Where(x => x.Expiry == expiry).ToList();

                // Calculate delta and gamma per strike price group for later filtering, ensure both call and put
available for option strategy
                var deltaGammaSymbols = new List<(decimal, decimal, Symbol, Symbol)>();
                var strikes = contracts.Select(x => x.Strike)
                    .Where(x => contracts.Count(y => y.Strike == x) == 2)
                    .Distinct();
                foreach (var strike in strikes)
                {
                    // Get both call and put for their aggregated delta and gamma
                    var call = contracts.Single(x => x.Right == OptionRight.Call && x.Strike == strike);
                    var put = contracts.Single(x => x.Right == OptionRight.Put && x.Strike == strike);
                    deltaGammaSymbols.Add((call.Greeks.Delta + put.Greeks.Delta, call.Greeks.Gamma +
put.Greeks.Gamma, call.Symbol, put.Symbol));
                }

                if (deltaGammaSymbols.Count == 0)
                {
                    continue;
                }

                // We want a delta-neutral position, so it is likely to be ATM (like a long straddle)
                // Less than 2d.p. difference is non-significant, which we can pick for better reward
```

```
                // Less than 2d.p. difference is non-significant, which we can risk for better reward
                // Assuming the market direction is up in most scenario, we try to get the strike providing max
overall Gamma
                // Make sure the aggregated gamma to be positive to bet on large uptrend
                // So it will earn more when the price really go up higher and higher, but locally immune for small
noise
                var filtered = deltaGammaSymbols.Where(item => Math.Round(item.Item1, 2) <= 0.01m && item.Item2 >
0).ToList();
                if (filtered.Count == 0)
                {
                    continue;
                }
                var selected = filtered.OrderByDescending(item => item.Item2).First();

                // Provide trade signal and roll the day count to let the position stay for the whole day
                var selectedCall = selected.Item3;
                var selectedPut = selected.Item4;
                insights.AddRange(new[] {
                    Insight.Price(selectedCall, Expiry.EndOfDay, InsightDirection.Up),
                    Insight.Price(selectedPut, Expiry.EndOfDay, InsightDirection.Up)
                });
                _day = slice.Time.Day;
            }
        }

        return insights;
    }
}

class SingleSharePortfolioConstructionModel : PortfolioConstructionModel
{
    public override IEnumerable<PortfolioTarget> CreateTargets(QCAlgorithm algorithm, Insight[] insights)
    {
        var targets = new List<PortfolioTarget>();
        foreach (var insight in insights)
        {
            if (algorithm.Securities[insight.Symbol].IsTradable)
            {
                // Use a whole number target to order the exact number of share for size-matching
                targets.Add(new PortfolioTarget(insight.Symbol, (int) insight.Direction));
            }
        }
        return targets;
    }
}
```

Data Point Attributes

The US Equity Options Universe dataset provides `OptionFilterUniverse` , `OptionUniverse` , and `OptionChain` objects.

OptionFilterUniverse Attributes

`OptionFilterUniverse` objects have the following attributes:

OptionUniverse Attributes

`OptionUniverse` objects have the following attributes:

OptionChain Attributes

`OptionChain` objects have the following attributes:

QuantConnect

US Equity Security Master

Introduction

The US Equity Security Master dataset by QuantConnect tracks US Equity corporate actions, including splits, dividends, delistings, mergers, and ticker changes through history. The data covers approximately 27,500 US Equities, starts in January 1998, and is delivered on a daily update frequency. You can easily download and install the dataset with the LEAN CLI so it's ready to use by LEAN. LEAN automatically handles all corporate actions and passes them into your algorithm as events .

This is `NOT` the underlying Equity data (US Equities dataset), which you need to purchase separately with a license from AlgoSeek. This security master dataset is required to purchase the US Equities or US Equities Options datasets.

For more information about the US Equity Security Master dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Data Summary

Data is delivered as a daily updated zip archive of map and factor files. The data is designed to be used in the LEAN Engine and cannot be consumed another way. The following table shows the dataset properties:

Property	**Value**
Start Date	January 1998
Data Points	Splits, Dividends, Mergers, IPO, & Delistings
Asset Coverage	27,500 US Equities
Resolution	Daily
Timezone	New York

This is `not` the underlying Equity data (US Equities dataset), which you need to purchase separately with a license from AlgoSeek. This security master dataset is required to purchase the US Equities or US Equities Options datasets.

Getting Started

You don't need any special code to utilize the US Equity Security Master. It automatically loads when you request US Equities data .

Accessing Splits

To get the current split data, index the `Splits` property of the current `Slice` with the Equity `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid

issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    // Check if any splits for the symbol
    if (slice.Splits.ContainsKey(_symbol))
    {
        // If so, get the mapped split object
        var split = slice.Splits[_symbol];
        Log($"Split: {split.Symbol}\t{split.SplitFactor}\t{split.ReferencePrice}\t{split.Type}");
    }
}
```

For more information about accessing splits, see Splits .

Accessing Dividends

To get the current dividend data, index the `Dividends` property of the current `Slice` with the Equity `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    // Check if any dividend for the symbol
    if (slice.Dividends.ContainsKey(_symbol))
    {
        // If so, get the mapped dividend object
        var dividend = slice.Dividends[_symbol];
        Log($"Dividend: {dividend.Symbol}\t{dividend.Distribution}\t{dividend.ReferencePrice}");
    }
}
```

For more information about accessing dividends, see Dividends .

Accessing Delistings

To get the current Delistings data, index the `Delistings` property of the current `Slice` with the Equity `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    // Check if any delisting for the symbol
    if (slice.Delistings.ContainsKey(_symbol))
    {
        // If so, get the mapped delisting object
        var delisting = slice.Delistings[_symbol];
        Log($"Delistings: {delisting.Type}");
    }
}
```

For more information about accessing delistings, see Delistings .

Accessing Symbol Change Events

To get the current Symbol change events, index the `SymbolChangedEvents` property of the current `Slice` with the Equity `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```C#
public override void OnData(Slice slice){
    // Check if any symbol change event for the symbol
    if (slice.SymbolChangedEvents.ContainsKey(_symbol))
    {
        // If so, get the mapped SymbolChangeEvent object
        var symbolChangedEvent = slice.SymbolChangedEvents[_symbol];
        Log($"Symbol changed: {symbolChangedEvent.OldSymbol} -> {symbolChangedEvent.NewSymbol}");
    }
}
```

For more information about accessing Symbol change events, see Symbol Changes .

Historical Data

To get historical US Equity Security Master data, call the `History` method with the data type and the Equity `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
// Splits
var splitHistory = History<Split>(_symbol, TimeSpan.FromDays(5*365));

// Dividends
var dividendHistory = History<Dividend>(_symbol, TimeSpan.FromDays(5*365));

// Symbol Changes
var symbolChangeHistory = History<SymbolChangedEvent>(_symbol, TimeSpan.FromDays(5*365));

// Delistings
var delistingHistory = History<Delisting>(_symbol, TimeSpan.FromDays(5*365));
```

For more information about historical data, see History Requests .

Live Trading Considerations

In backtesting, corporate actions occurs at midnight (ET). In live trading, the live data for corporate actions arrives at 6/7 AM ET, so that's when they occur.

Supported Assets

To view the supported assets in the US Equity Security Master dataset, see the Data Explorer . This dataset doesn't include Over-the-Counter (OTC) stocks.

Example Applications

The US Security Master enables you to accurately design strategies harnessing any core corporate actions. Examples include the following strategies:

- Post-dividend announcement trading strategies.
- Trading on new Equities by monitoring for IPOs.
- Harnessing split announcements for reverse-split announcement momentum.

Classic Algorithm Example

The following example algorithm logs the `Split` , `Dividend` , `Delisting` , and `SymbolChangedEvent` objects of Apple:

```
public class USEquitySecurityMasterAlgorithm : QCAlgorithm { private Symbol _equity; public override void Initialize() { SetStartDate(2024, 9, 1); SetEndDate(2024, 12, 31); SetCash(1000000); _equity = AddEquity("AAPL", Resolution.Daily).Symbol; } public override void OnData(Slice slice) { // Accessing Data - Splits if (slice.Splits.ContainsKey(_equity)) { var split = slice.Splits[_equity]; Debug($"Split: {split.Symbol}\t{split.SplitFactor}\t{split.ReferencePrice}\t{split.Type}"); } // Accessing Data - Dividends if (slice.Dividends.ContainsKey(_equity)) { var dividend = slice.Dividends[_equity]; Log($"Dividend: {dividend.Symbol}\t{dividend.Distribution}\t{dividend.ReferencePrice}"); } // Accessing Data - Delisting if (slice.Delistings.ContainsKey(_equity)) { var delisting = slice.Delistings[_equity]; Log($"Delistings: {delisting.Type}"); } // Accessing Data - Symbol Changed Event if (slice.SymbolChangedEvents.ContainsKey(_equity)) { var symbolChangedEvent = slice.SymbolChangedEvents[_equity]; Log($"Symbol changed: {symbolChangedEvent.OldSymbol} -> {symbolChangedEvent.NewSymbol}"); } } }
```

Framework Algorithm Example

The following algorithm demonstrates the payments for cash dividends in backtesting. When the data normalization mode is `Raw`, your portfolio receives cash dividends.

C#

```
public class PaymentsAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // this will use the Tradier Brokerage open order split behavior
        // forward split will modify open order to maintain order value
        // reverse split open orders will be canceled
        SetBrokerageModel(BrokerageName.TradierBrokerage);

        UniverseSettings.Resolution = Resolution.Daily;
        UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;

        // MSFT: Splits and Dividends
        // GOOG: Symbol Changed Event
        // AAA.1: Delisting
        SetUniverseSelection(new ManualUniverseSelectionModel(
            QuantConnect.Symbol.Create("MSFT", SecurityType.Equity, Market.USA)));

        SetAlpha(new PaymentAlphaModel());
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
        SetExecution(new BracketExecutionModel());
    }
}

public class PaymentAlphaModel : AlphaModel
{
    private Symbol _symbol = Symbol.Empty;

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        // Accessing Data - Splits
        if (slice.Splits.ContainsKey(_symbol))
        {
            var split = slice.Splits[_symbol];
            algorithm.Debug($"{split.Time.ToIso8601Invariant()} >> SPLIT >> {split.Symbol} - " +
                $"{split.SplitFactor.ToStringInvariant()} - " +
                $"{algorithm.Portfolio.Cash.ToStringInvariant()} - " +
                $"{algorithm.Portfolio[_symbol].Quantity.ToStringInvariant()}");
        }

        // Accessing Data - Dividends
        if (slice.Dividends.ContainsKey(_symbol))
        {
            var dividend = slice.Dividends[_symbol];
            algorithm.Debug($"{dividend.Time.ToStringInvariant("o")} >> DIVIDEND >> {dividend.Symbol} - " +
                $"{dividend.Distribution.ToStringInvariant("C")} - {algorithm.Portfolio.Cash} - " +
                $"{algorithm.Portfolio[_symbol].Price.ToStringInvariant("C")}");
        }

        // Accessing Data - Delisting
        if (slice.Delistings.ContainsKey(_symbol))
        {
```

```
            var delisting = slice.Delistings[_symbol];
            algorithm.Debug($"{delisting.Time.ToStringInvariant("o")} >> DELISTING >> {delisting.Type}");
        }

        // Accessing Data - Symbol Changed Event
        if (slice.SymbolChangedEvents.ContainsKey(_symbol))
        {
            var symbolChangedEvent = slice.SymbolChangedEvents[_symbol];
            algorithm.Debug($"{symbolChangedEvent.Time.ToStringInvariant("o")} >> Symbol Changed Event >> " +
                $"{symbolChangedEvent.OldSymbol} -> {symbolChangedEvent.OldSymbol}");
        }

        return slice.Bars.ContainsKey(_symbol)
            ? new [] { Insight.Price(_symbol, TimeSpan.FromDays(1), InsightDirection.Up) }
            : Enumerable.Empty<Insight>();
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        _symbol = changes.AddedSecurities.First().Symbol;
    }
}

public class BracketExecutionModel : ExecutionModel
{
    private readonly PortfolioTargetCollection _targetsCollection = new PortfolioTargetCollection();

    public override void Execute(QCAlgorithm algorithm, IPortfolioTarget[] targets)
    {
        _targetsCollection.AddRange(targets);
        // for performance we check count value, OrderByMarginImpact and ClearFulfilled are expensive to call
        if (_targetsCollection.Count > 0)
        {
            foreach (var target in _targetsCollection.OrderByMarginImpact(algorithm))
            {
                // calculate remaining quantity to be ordered
                var quantity = OrderSizing.GetUnorderedQuantity(algorithm, target);
                if (quantity != 0 && algorithm.Transactions.OrdersCount == 0)
                {
                    var bar = algorithm.Securities[target.Symbol].GetLastData() as TradeBar;
                    algorithm.MarketOrder(target.Symbol, quantity);
                    // place some orders that won't fill, when the split comes in they'll get modified to reflect the
split
                    algorithm.StopMarketOrder(target.Symbol, -quantity, bar.Low/2);
                    algorithm.LimitOrder(target.Symbol, -quantity, bar.High*2);
                }
            }

            _targetsCollection.ClearFulfilled(algorithm);
        }
    }
}
```

Data Point Attributes

The US Equity Security Master dataset provides `Split` , `Dividend` , `Delisting` , and `SymbolChangedEvent` objects.

Split Attributes

When a split or merger occurs, we pass the previous `Symbol` data into your algorithm. `Split` objects have the following attributes:

Dividend Attributes

Dividend events are triggered on the payment date. `Dividend` objects have the following attributes:

Delisting Attributes

When a security is delisted, we notify your algorithm. `Delisting` objects have the following attributes:

SymbolChangedEvent Attributes

When a security changes their ticker, we notify your algorithm. `SymbolChangedEvent` objects have the following attributes:

QuantConnect

US Future Option Universe

Introduction

The US Future Option Universe dataset by QuantConnect lists the available US Future Options contracts and the current open interest. The data covers 15 Monthly Future contracts, starts in January 2012, and is delivered on a daily update frequency. This dataset is created by monitoring the trading activity on the CME, CBOT, NYMEX, and COMEX markets.

The US Future Option Universe dataset depends on the US Future Universe dataset because the US Future Universe dataset contains the universe of underlying Futures contracts. This dataset also depends on the US Futures Security Master dataset because the US Futures Security Master dataset contains information on symbol changes of the contracts.

This dataset `does not` contain market data. For market data, see US Future Options by AlgoSeek .

For more information about the US Future Option Universe dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Getting Started

The following snippet demonstrates how to request data from the US Future Options Universe dataset:

```C#
var future = AddFuture(Futures.Metals.Gold, Resolution.Minute);
future.SetFilter(0, 90);
AddFutureOption(future.Symbol, universe => universe.Strikes(-1, 1));
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2012
Asset Coverage	15 Monthly Future Contracts. Standard expires only. No weeklies or 0DTE contracts.
Data Density	Dense
Resolution	Daily
Timezone	New York
Market Hours	Regular and Extended

Requesting Data

To add US Future Options Universe data to your algorithm, call the `AddFutureOption` method. To define which contracts should be in your universe, specify the filter when requesting the Future Option data.

The `AddFutureOption` method provides a daily stream of Option chain data. To get the most recent daily chain, call the `OptionChain` method with the underlying Future Symbol. The `OptionChain` method returns data on all the tradable contracts, not just the contracts that pass your universe filter.

```
C#
public class USFutureOptionsDataAlgorithm : QCAlgorithm
{
    private Future _future;

    public override void Initialize()
    {
        SetStartDate(2020, 6, 1);
        SetEndDate(2021, 6, 1);
        SetCash(100000);
        UniverseSettings.Asynchronous = true;

        _future = AddFuture(Futures.Metals.Gold, Resolution.Minute);
        _future.SetFilter(0, 90);
        // Set our strike/expiry filter for this option chain
        AddFutureOption(_future.Symbol, OptionFilter);
    }

    public override void OnData(Slice slice)
    {
        // Create canonical symbol for the mapped future contract, since option chains are mapped by canonical
symbol.
        var symbol = QuantConnect.Symbol.CreateCanonicalOption(_future.Mapped);
        // Get the entire Option chain for the current day.
        var chain = OptionChain(symbol);
    }

    private OptionFilterUniverse OptionFilter(OptionFilterUniverse universe)
    {
        // Contracts can be filtered by strike, and expiration
        return universe.Strikes(-1, 1);
    }
}
```

The Future resolution must be less than or equal to the Future Option resolution. For example, if you set the Future resolution to minute, then you must set the Future Option resolution to minute, hour, or daily.

For more information about creating US Future Option universes, see Future Options .

Accessing Data

For information about accessing US Future Options Universe data, see Future Options .

Historical Data

You can get historical US Future Options Universe data in an algorithm and the Research Environment.

Historical Data In Algorithms

To get historical US Future Options Universe data in an algorithm, call the `History<OptionUniverse>` method with the canonical mapped Future Option `Symbol` . This method returns data on all of the tradable contracts, not just the contracts that pass your universe filter. If there is no data in the period you request, the history result is empty.

```
C#
var futureOptionSymbol = QuantConnect.Symbol.CreateCanonicalOption(_future.Mapped);
var history = History<OptionUniverse>(futureOptionSymbol, TimeSpan.FromDays(10)).ToList();
```

For more information about historical US Future Options Universe data in algorithms, see Historical Data .

Historical Data In Research

To get historical US Future Options Universe data in the Research Environment, call the `History<OptionUniverse>` method with the canonical Option `Symbol` . This method returns data on all of the tradable contracts, not just the contracts that pass your universe filter.

```
C#
var qb = new QuantBook();
var future = qb.AddFuture(Futures.Metals.Gold, Resolution.Minute);
var symbol = QuantConnect.Symbol.CreateCanonicalOption(future.Mapped);
var history = qb.History<OptionUniverse>(symbol, new DateTime(2020, 6, 1), new DateTime(2020, 6, 6));
foreach (var chain in history)
{
    var endTime = chain.EndTime;
    var filteredContracts = chain.Data
        .Select(contract => contract as OptionUniverse)
        .Where(contract => contract.Greeks.Delta > 0.3m);
    foreach (var contract in filteredContracts)
    {
        var price = contract.Price;
        var iv = contract.ImpliedVolatility;
    }
}
```

For more information about historical Future Options Universe data in the Research Environment, see Universes .

Supported Assets

The following list shows the available (15) Futures Options:

- `Futures.Energy.CrudeOilWTI` : Crude Oil WTI Futures (NYMEX: LO | Underlying: CL)
- `Futures.Energy.Gasoline` : Gasoline RBOB Futures (NYMEX: OB | Underlying: RB)
- `Futures.Energy.HeatingOil` : Heating Oil Futures (NYMEX: OH | Underlying: HO)
- `Futures.Energy.NaturalGas` : Natural Gas Futures (NYMEX: ON | Underlying: NG)
- `Futures.Financials.Y10TreasuryNote` : 10Y U.S. Treasury Note Futures (CBOT: OZN | Underlying: ZN)
- `Futures.Financials.Y2TreasuryNote` : 2Y U.S. Treasury Note Futures (CBOT: OZT | Underlying: ZT)
- `Futures.Financials.Y30TreasuryBond` : 30Y U.S. Treasury Bond Futures (CBOT: OZB | Underlying: ZB)
- `Futures.Grains.Corn` : Corn Futures (CBOT: OZC | Underlying: ZC)
- `Futures.Grains.Soybeans` : Soybeans Futures (CBOT: OZS | Underlying: ZS)
- `Futures.Grains.Wheat` : Default wheat contract is SRWWheat (CBOT: OZW | Underlying: ZW)
- `Futures.Indices.NASDAQ100EMini` : E-mini NASDAQ 100 Futures (CME: NQ)
- `Futures.Indices.SP500EMini` : E-mini S&P 500 Futures (CME: ES)
- `Futures.Metals.Copper` : Copper Futures (COMEX: HXE | Underlying: HG)
- `Futures.Metals.Gold` : Gold Futures (COMEX: OG | Underlying: GC)
- `Futures.Metals.Silver` : Silver Futures (COMEX: SO | Underlying: SI)

Example Applications

The US Future Options dataset enables you to accurately design Future Option strategies. Examples include the following strategies:

- Selling out of the money Future Option contracts to collect the premium that the Option buyer pays
- Buying put Options to hedge against downward price movement in Future contracts you bought
- Exploiting arbitrage opportunities that arise when the price of Option contracts deviate from their theoretical value

Classic Algorithm Example

The following example demonstrates a bull call spread Option strategy using universe filtering.

```C#
public class FutureOptionAlgorithm : QCAlgorithm
{
    private Future _underlying;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Filter the underlying continuous Futures to narrow the FOP spectrum.
        _underlying = AddFuture(Futures.Indices.SP500EMini,
            extendedMarketHours: true,
            dataMappingMode: DataMappingMode.OpenInterest,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            contractDepthOffset: 0);
        _underlying.SetFilter(0, 182);
        // Use CallSpread filter to obtain the 2 best-matched contracts that forms a call spread.
        // It simplifies from further filtering and reduce computation on redundant subscription.
        AddFutureOption(_underlying.Symbol, (u) => u.CallSpread(5, 5, -5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested)
            return;

        // Create canonical symbol for the mapped future contract, since we need that to access the option chain.
        var symbol = QuantConnect.Symbol.CreateCanonicalOption(_underlying.Mapped);

        // Get option chain data for the mapped future only.
        // It requires 2 contracts with different strikes to form a call spread, so we make sure at least 2 contracts
are present.
        if (!slice.OptionChains.TryGetValue(symbol, out var chain) || chain.Count() < 2)
            return;

        // Separate the contracts by strike, as we need to access their strike.
        var expiry = chain.Min(x => x.Expiry);
        var itmStrike = chain.Min(x => x.Strike);
        var otmStrike = chain.Max(x => x.Strike);

        // Use abstraction method to order a bull call spread to avoid manual error.
        var optionStrategy = OptionStrategies.BullCallSpread(symbol, itmStrike, otmStrike, expiry);
        Buy(optionStrategy, 1);
    }
}
```

Data Point Attributes

The US Future Options Universe dataset provides `OptionFilterUniverse` and `OptionUniverse` objects.

OptionFilterUniverse Attributes

`OptionFilterUniverse` objects have the following attributes:

OptionUniverse Attributes

`OptionUniverse` objects have the following attributes:

QuantConnect

US Future Universe

Introduction

The US Future Universe dataset by QuantConnect lists the available US Future contracts, their daily trading volume, and Open Interest. The data covers the 157 most liquid contracts, starts in May 2009, and is delivered on daily frequency. This dataset is created by monitoring the trading activity on the CFE, CBOT, CME, COMEX, NYMEX, and ICE * .

This dataset depends on the US Futures Security Master dataset because the US Futures Security Master dataset contains information on symbol changes of the contracts.

This dataset `does not` contain market data. For market data, see US Futures by AlgoSeek .

For more information about the US Future Universe dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Getting Started

The following snippet demonstrates how to request data from the US Future Universe dataset:

```C#
var future = AddFuture(Futures.Metals.Gold);
future.SetFilter(0, 90);
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	May 2009
Asset Coverage	157 Futures
Data Density	Dense
Resolution	Daily
Timezone	• Chicago (CFE, CME or CBOT) • New York (ICE, NYMEX or COMEX)
Market Hours	Regular and Extended

Requesting Data

To add US Future Universe data to your algorithm, call the `AddFuture` method. Save a reference to the Future object so you can access the data later in your algorithm. To define which contracts should be in your universe, specify the filter when requesting the Future data.

The `AddFuture` method provides a daily stream of Future chain data. To get the most recent daily chain, call the `FuturesChain` method with the underlying Future Symbol. The `FuturesChain` method returns data on all the tradable contracts, not just the contracts that pass your universe filter.

```
C#

public class USFutureDataAlgorithm : QCAlgorithm
{
    private Future _future;

    public override void Initialize()
    {
        SetStartDate(2020, 6, 1);
        SetEndDate(2021, 6, 1);
        SetCash(100000);
        UniverseSettings.Asynchronous = true;
        _future = AddFuture(Futures.Metals.Gold);
        // Set our contract filter for this Future chain.
        _future.SetFilter((universe) => universe.StandardsOnly().FrontMonth());
        // Get the entire Futures chain for the current day.
        var chain = FuturesChain(_future.Symbol);
    }
}
```

For more information about creating US Future universes, see Futures .

Accessing Data

For information about accessing US Future Universe data, see Futures .

Historical Data

You can get historical US Future Universe data in an algorithm and the Research Environment.

Historical Data In Algorithms

To get historical US Future Universe data in an algorithm, call the `History<FutureUniverse>` method with continuous Future `Symbol` and a lookback period. This method returns the all the available contracts for each trading day, not the subset of contracts that pass your universe filter. If there is no data for the period you requested, the history result is empty.

```
C#

// Add the Future and save a reference to it.
var future = AddFuture(Futures.Metals.Gold);

// Get historical data.
var history = History<FutureUniverse>(future.Symbol, 5);
```

For more information about historical US Future Universe data in algorithms, see Contracts .

Historical Data In Research

To get historical US Future Universe data in the Research Environment, call the `History<FutureUniverse>` method with continuous Future `Symbol` and a time period. This method returns the all the available contracts for each trading day, not the subset of contracts that pass your universe filter. If there is no data for the period you requested, the history result is empty.

```
// Add the Future and save a reference to it.
var future = qb.AddFuture(Futures.Metals.Gold);

// Get historical data.
var history = qb.History<FutureUniverse>(future.Symbol, new DateTime(2025, 1, 1), new DateTime(2025, 4, 1));
```

For more information about historical US Future Universe data in the Research Environment, see Daily Prices History .

Supported Assets

The following list shows the available (157) Futures:

- `Futures.Currencies.AUD` : Australian Dollar Futures (CME: 6A)
- `Futures.Currencies.AUDCAD` : Australian Dollar/Canadian Dollar Futures (CME: ACD)
- `Futures.Currencies.AUDJPY` : Australian Dollar/Japanese Yen Futures (CME: AJY)
- `Futures.Currencies.AUDNZD` : Australian Dollar/New Zealand Dollar Futures (CME: ANE)
- `Futures.Currencies.BRL` : Brazillian Real Futures (CME: 6L)
- `Futures.Currencies.BTC` : Bitcoin Futures (CME: BTC)
- `Futures.Currencies.BTICMicroBTC` : BTIC on Micro Bitcoin Futures (CME: MIB)
- `Futures.Currencies.BTICMicroEther` : BTIC on Micro Ether Futures (CME: MRB)
- `Futures.Currencies.CAD` : Canadian Dollar Futures (CME: 6C)
- `Futures.Currencies.CADJPY` : Canadian Dollar/Japanese Yen Futures (CME: CJY)
- `Futures.Currencies.CHF` : Swiss Franc Futures (CME: 6S)
- `Futures.Currencies.ETH` : Ether Futures (CME: ETH)
- `Futures.Currencies.EUR` : Euro FX Futures (CME: 6E)
- `Futures.Currencies.EURAUD` : Euro/Australian Dollar Futures (CME: EAD)
- `Futures.Currencies.EURCAD` : Euro/Canadian Dollar Futures (CME: ECD)
- `Futures.Currencies.EuroFXEmini` : E-mini Euro FX Futures (CME: E7)
- `Futures.Currencies.EURSEK` : Euro/Swedish Krona Futures (CME: ESK)
- `Futures.Currencies.GBP` : British Pound Futures (CME: 6B)
- `Futures.Currencies.JapaneseYenEmini` : E-mini Japanese Yen Futures (CME: J7)
- `Futures.Currencies.JPY` : Japanese Yen Futures (CME: 6J)
- `Futures.Currencies.MicroAUD` : Micro AUD/USD Futures (CME: M6A)
- `Futures.Currencies.MicroBTC` : Micro Bitcoin Futures (CME: MBT)
- `Futures.Currencies.MicroCAD` : Micro USD/CAD Futures (CME: M6C)
- `Futures.Currencies.MicroCADUSD` : Micro CAD/USD Futures (CME: MCD)
- `Futures.Currencies.MicroCHF` : Micro CHF/USD Futures (CME: MSF)
- `Futures.Currencies.MicroEther` : Micro Ether Futures (CME: MET)
- `Futures.Currencies.MicroEUR` : Micro EUR/USD Futures (CME: M6E)
- `Futures.Currencies.MicroGBP` : Micro GBP/USD Futures (CME: M6B)
- `Futures.Currencies.MicroINRUSD` : Micro INR/USD Futures (CME: MIR)
- `Futures.Currencies.MicroJPY` : Micro JPY/USD Futures (CME: MJY)
- `Futures.Currencies.MicroUSDCHF` : Micro USD/CHF Futures (CME: M6S)
- `Futures.Currencies.MicroUSDCNH` : Micro USD/CNH Futures (CME: MNH)
- `Futures.Currencies.MicroUSDJPY` : Micro USD/JPY Futures (CME: M6J)
- `Futures.Currencies.MXN` : Mexican Peso Futures (CME: 6M)
- `Futures.Currencies.NZD` : New Zealand Dollar Futures (CME: 6N)

- `Futures.Currencies.RUB` : Russian Ruble Futures (CME: 6R)
- `Futures.Currencies.StandardSizeUSDOffshoreRMBCNH` : Standard-Size USD/Offshore RMB (CNH) Futures (CME: CNH)
- `Futures.Currencies.ZAR` : South African Rand Futures (CME: 6Z)
- `Futures.Energy.ArgusLLSvsWTIArgusTradeMonth` : Argus LLS vs. WTI (Argus) Trade Month Futures (NYMEX: AE5)
- `Futures.Energy.ArgusPropaneFarEastIndex` : Argus Propane Far East Index Futures (NYMEX: A7E)
- `Futures.Energy.ArgusPropaneSaudiAramco` : Argus Propane (Saudi Aramco) Futures (NYMEX: A9N)
- `Futures.Energy.BrentCrudeOilVsDubaiCrudeOilPlatts` : Brent Crude Oil vs. Dubai Crude Oil (Platts) Futures (NYMEX: ADB)
- `Futures.Energy.BrentLastDayFinancial` : Brent Last Day Financial Futures (NYMEX: BZ)
- `Futures.Energy.ChicagoEthanolPlatts` : Chicago Ethaanol (Platts) Futures (NYMEX: CU)
- `Futures.Energy.ConwayPropaneOPIS` : Conway Propane (OPIS) Futures (NYMEX: A8K)
- `Futures.Energy.CrudeOilWTI` : Crude Oil WTI Futures (NYMEX: CL)
- `Futures.Energy.DubaiCrudeOilPlattsFinancial` : Dubai Crude Oil (Platts) Financial Futures (NYMEX: DCB)
- `Futures.Energy.EastWestGasolineSpreadPlattsArgus` : East-West Gasoline Spread (Platts-Argus) Futures (NYMEX: EWG)
- `Futures.Energy.EastWestNaphthaJapanCFvsCargoesCIFNWESpreadPlatts` : East-West Naphtha: Japan C&F vs. Cargoes CIF NWE Spread (Platts) Futures (NYMEX: EWN)
- `Futures.Energy.Ethanol` : Ethanol Futures (CBOT: EH)
- `Futures.Energy.EthanolT2FOBRdamIncludingDutyPlatts` : Ethanol T2 FOB Rdam Including Duty (Platts) Futures (NYMEX: AZ1)
- `Futures.Energy.EuropeanNaphthaPlattsCrackSpread` : European Naphtha (Platts) Crack Spread Futures (NYMEX: EN)
- `Futures.Energy.EuropeanPropaneCIFARAArgus` : European Propane CIF ARA (Argus) Futures (NYMEX: APS)
- `Futures.Energy.EuropeanPropaneCIFARAArgusVsNaphthaCargoesCIFNWEPlatts` : European Propane CIF ARA (Argus) vs. Naphtha Cargoes CIF NWE (Platts) Futures (NYMEX: EPN)
- `Futures.Energy.FreightRouteTC14Baltic` : Freight Route TC14 (Baltic) Futures (NYMEX: FRC)
- `Futures.Energy.Gasoline` : Gasoline RBOB Futures (NYMEX: RB)
- `Futures.Energy.GasolineEurobobOxyNWEBargesArgus` : Gasoline Euro-bob Oxy NWE Barges (Argus) Futures (NYMEX: B7H)
- `Futures.Energy.GroupThreeSuboctaneGasolinePlattsVsRBOB` : Group Three Sub-octane Gasoliine (Platts) vs. RBOB Futures (NYMEX: AA8)
- `Futures.Energy.GroupThreeULSDPlattsVsNYHarborULSD` : Group Three ULSD (Platts) vs. NY Harbor ULSD Futures (NYMEX: AA6)
- `Futures.Energy.GulfCoastCBOBGasolineA2PlattsVsRBOBGasoline` : Gulf Coast CBOB Gasoline A2 (Platts) vs. RBOB Gasoline Futures (NYMEX: CRB)
- `Futures.Energy.GulfCoastHSFOPlattsVsEuropeanThreePointFivePercentFuelOilBargesFOBRdamPlatts` : Gulf Coast HSFO (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures (NYMEX: GCU)
- `Futures.Energy.HeatingOil` : Heating Oil Futures (NYMEX: HO)
- `Futures.Energy.LosAngelesCARBOBGasolineOPISvsRBOBGasoline` : Los Angeles CARBOB Gasoline (OPIS) vs. RBOB Gasoline Futures (NYMEX: AJL)
- `Futures.Energy.LosAngelesCARBDieselOPISvsNYHarborULSD` : Los Angeles CARB Diesel (OPIS) vs. NY Harbor ULSD Futures (NYMEX: AKL)
- `Futures.Energy.LosAngelesJetOPISvsNYHarborULSD` : Los Angeles Jet (OPIS) vs. NY Harbor ULSD Futures (NYMEX: AJS)
- `Futures.Energy.MarsArgusVsWTIFinancial` : Mars (Argus) vs. WTI Financial Futures (NYMEX: AYX)
- `Futures.Energy.MarsArgusVsWTITradeMonth` : Mars (Argus) vs. WTI Trade Month Futures (NYMEX: AYV)
- `Futures.Energy.MicroCrudeOilWTI` : Micro WTI Crude Oil Futures (NYMEX: MCL)

- `Futures.Energy.MicroEuropeanFOBRdamMarineFuelZeroPointFivePercentBargesPlatts` : Micro European FOB Rdam Marine Fuel 0.5% Barges (Platts) Futures (NYMEX: R5O)
- `Futures.Energy.MicroEuropeanThreePointFivePercentOilBargesFOBRdamPlatts` : Micro European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures (NYMEX: MEF)
- `Futures.Energy.MicroGasoilZeroPointOnePercentBargesFOBARAPlatts` : Micro Gasoil 0.1% Barges FOB ARA (Platts) Futures (NYMEX: M1B)
- `Futures.Energy.MicroSingaporeFOBMarineFuelZeroPointFivePercetPlatts` : Micro Singapore FOB Marine Fuel 0.5% (Platts) Futures (NYMEX: S5O)
- `Futures.Energy.MicroSingaporeFuelOil380CSTPlatts` : Micro Singapore Fuel Oil 380CST (Platts) Futures (NYMEX: MAF)
- `Futures.Energy.MiniEuropeanThreePointPercentFiveFuelOilBargesPlatts` : Mini European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures (NYMEX: A0D)
- `Futures.Energy.MiniSingaporeFuelOil180CstPlatts` : Mini Singapore Fuel Oil 180 cst (Platts) Futures (NYMEX: A0F)
- `Futures.Energy.MontBelvieuEthaneOPIS` : Mont Belvieu Ethane (OPIS) Futures (NYMEX: AC0)
- `Futures.Energy.MontBelvieuLDHPropaneOPIS` : Mont Belvieu LDH Propane (OPIS) Futures (NYMEX: B0)
- `Futures.Energy.MontBelvieuNaturalGasolineOPIS` : Mont Belvieu Natural Gasoline (OPIS) Futures (NYMEX: A7Q)
- `Futures.Energy.MontBelvieuNormalButaneOPIS` : Mont Belvieu Normal Butane (OPIS) Futures (NYMEX: AD0)
- `Futures.Energy.NaturalGas` : Natural Gas Futures (NYMEX: NG)
- `Futures.Energy.NaturalGasHenryHubLastDayFinancial` : Natural Gas (Henry Hub) Last-day Financial Futures (NYMEX: HH)
- `Futures.Energy.NaturalGasHenryHubPenultimateFinancial` : Natural Gas (Henry Hub) Penultimate Financial Futures (NYMEX: HP)
- `Futures.Energy.OnePercentFuelOilCargoesFOBNWEPlattsVsThreePointFivePercentFuelOilBargesFOBRdamPlatts` : 1% Fuel Oil Cargoes FOB NWE (Platts) vs. 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures (NYMEX: FSS)
- `Futures.Energy.PremiumUnleadedGasoline10ppmFOBMEDPlatts` : Premium Unleaded Gasoline 10 ppm FOB MED (Platts) Futures (NYMEX: A3G)
- `Futures.Energy.PropaneNonLDHMontBelvieuOPIS` : Propane Non-LDH Mont Belvieu (OPIS) Futures (NYMEX: A1R)
- `Futures.Energy.RBOBGasolineCrackSpread` : RBOB Gasoline Crack Spread Futures (NYMEX: ARE)
- `Futures.Energy.RBOBGasolineVsEurobobOxyNWEBargesArgusThreeHundredFiftyThousandGallons` : RBOB Gasoline vs. Euro-bob Oxy NWE Barges (Argus) (350,000 gallons) Futures (NYMEX: EXR)
- `Futures.Energy.SingaporeFuelOil380cstPlattsVsEuropeanThreePointFivePercentFuelOilBargesFOBRdamPlatts` : Singapore Fuel Oil 380 cst (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures (NYMEX: EVC)
- `Futures.Energy.SingaporeGasoilPlattsVsLowSulphurGasoilFutures` : Singapore Gasoil (Platts) vs. Low Sulphur Gasoil Futures (NYMEX: AGA)
- `Futures.Energy.SingaporeMogas92UnleadedPlattsBrentCrackSpread` : Singapore Mogas 92 Unleaded (Platts) Brent Crack Spread Futures (NYMEX: D1N)
- `Futures.Energy.ThreePointFivePercentFuelOilBargesFOBRdamPlattsCrackSpread` : 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread Futures (NYMEX: FO)
- `Futures.Energy.ThreePointFivePercentFuelOilBargesFOBRdamPlattsCrackSpread1000mt` : 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread (1000mt) Futures (NYMEX: BOO)
- `Futures.Energy.WTIBrentFinancial` : WTI-Brent Financial Futures (NYMEX: BK)
- `Futures.Energy.WTIFinancial` : WTI Financial Futures (NYMEX: CSX)
- `Futures.Energy.WTIHoustonArgusVsWTITradeMonth` : WTI Houston (Argus) vs. WTI Trade Month Futures (NYMEX: HTT)
- `Futures.Energy.WTIHoustonCrudeOil` : WTI Houston Crude Oil Futures (NYMEX: HCL)
- `Futures.Financials.EuroDollar` : EuroDollar Futures (CME: GE)
- `Futures.Financials.FiveYearUSDMACSwap` : 5-Year USD MAC Swap Futures (CBOT: F1U)

- `Futures.Financials.MicroY10TreasuryNote` : Micro 10-Year Yield Futures (CBOT: 10Y)
- `Futures.Financials.MicroY2TreasuryBond` : Micro 2-Year Yield Futures (CBOT: 2YY)
- `Futures.Financials.MicroY30TreasuryBond` : Micro 30-Year Yield Futures (CBOT: 30Y)
- `Futures.Financials.MicroY5TreasuryBond` : Micro 5-Year Yield Futures (CBOT: 5YY)
- `Futures.Financials.UltraTenYearUSTreasuryNote` : Ultra 10-Year U.S. Treasury Note Futures (CBOT: TN)
- `Futures.Financials.UltraUSTreasuryBond` : Ultra U.S. Treasury Bond Futures (CBOT: UB)
- `Futures.Financials.Y10TreasuryNote` : 10Y U.S. Treasury Note Futures (CBOT: ZN)
- `Futures.Financials.Y2TreasuryNote` : 2Y U.S. Treasury Note Futures (CBOT: ZT)
- `Futures.Financials.Y30TreasuryBond` : 30Y U.S. Treasury Bond Futures (CBOT: ZB)
- `Futures.Financials.Y5TreasuryNote` : 5Y U.S. Treasury Note Futures (CBOT: ZF)
- `Futures.Forestry.Lumber` : Lumber Futures (CME: LBR)
- `Futures.Forestry.RandomLengthLumber` : Random Length Lumber Futures (CME: LBS)
- `Futures.Grains.BlackSeaCornFinanciallySettledPlatts` : Black Sea Corn Financially Settled (Platts) Futures (CBOT: BCF)
- `Futures.Grains.BlackSeaWheatFinanciallySettledPlatts` : Black Sea Wheat Financially Settled (Platts) Futures (CBOT: BWF)
- `Futures.Grains.Corn` : Corn Futures (CBOT: ZC)
- `Futures.Grains.HRWWheat` : KC HRW Wheat Futures (CBOT: KE)
- `Futures.Grains.Oats` : Oats Futures (CBOT: ZO)
- `Futures.Grains.Soybeans` : Soybeans Futures (CBOT: ZS)
- `Futures.Grains.SoybeanMeal` : Soybean Meal Futures (CBOT: ZM)
- `Futures.Grains.SoybeanOil` : Soybean Oil Futures (CBOT: ZL)
- `Futures.Grains.Wheat` : Default wheat contract is SRWWheat (CBOT: ZW)
- `Futures.Indices.BloombergCommodityIndex` : Bloomberg Commodity Index Futures (CBOT: AW)
- `Futures.Indices.Dow30EMini` : E-mini Dow Indu 30 Futures (CBOT: YM)
- `Futures.Indices.DowJonesRealEstate` : Dow Jones Real Estate futures on CME (CME: RX)
- `Futures.Indices.FTSEEmergingEmini` : E-mini FTSE Emerging Index Futures (CME: EI)
- `Futures.Indices.MicroDow30EMini` : Micro E-mini Dow Jones Industrial Average Index Futures (CBOT: MYM)
- `Futures.Indices.MicroNASDAQ100EMini` : Micro E-mini Nasdaq-100 Index Futures (CME: MNQ)
- `Futures.Indices.MicroRussell2000EMini` : Micro E-mini Russell 2000 Index Futures (CME: M2K)
- `Futures.Indices.MicroSP500EMini` : Micro E-mini S&P 500 Index Futures (CME: MES)
- `Futures.Indices.NASDAQ100BiotechnologyEMini` : E-mini Nasdaq-100 Biotechnology Index Futures (CME: BIO)
- `Futures.Indices.NASDAQ100EMini` : E-mini NASDAQ 100 Futures (CME: NQ)
- `Futures.Indices.Nikkei225Dollar` : Nikkei-225 Dollar Futures (CME: NKD)
- `Futures.Indices.Nikkei225YenCME` : Nikkei-225 Yen denominated Futures on CME (CME: NIY)
- `Futures.Indices.Russell1000EMini` : E-mini Russell 1000 futures on CME (CME: RS1)
- `Futures.Indices.Russell2000EMini` : E-mini Russell 2000 Futures (CME: RTY)
- `Futures.Indices.SPGSCICommodity` : S&P-GSCI Commodity Index Futures (CME: GD)
- `Futures.Indices.SP400MidCapEmini` : E-mini S&P MidCap 400 Futures (CME: EMD)
- `Futures.Indices.SP500AnnualDividendIndex` : (CME: SDA)
- `Futures.Indices.SP500EMini` : E-mini S&P 500 Futures (CME: ES)
- `Futures.Indices.TOPIXYEN` : YEN Denominated Topix Index Futures on CME (CME: TPY)
- `Futures.Indices.USDDenominatedIbovespa` : USD-Denominated Ibovespa Index Futures (CME: IBV)
- `Futures.Indices.VIX` : CBOE Volatility Index Futures (CFE: VX)
- `Futures.Meats.FeederCattle` : Feeder Cattle Futures (CME: GF)

- `Futures.Meats.LeanHogs` : Lean Hogs Futures (CME: HE)
- `Futures.Meats.LiveCattle` : Live Cattle Futures (CME: LE)
- `Futures.Metals.AluminiumEuropeanPremiumDutyPaidMetalBulletin` : Aluminium European Premium Duty-Paid (Metal Bulletin) Futures (COMEX: EDP)
- `Futures.Metals.AluminumMWUSTransactionPremiumPlatts25MT` : Aluminum MW U.S. Transaction Premium Platts (25MT) Futures (COMEX: AUP)
- `Futures.Metals.Copper` : Copper Futures (COMEX: HG)
- `Futures.Metals.Gold` : Gold Futures (COMEX: GC)
- `Futures.Metals.MicroGold` : Micro Gold Futures (COMEX: MGC)
- `Futures.Metals.MicroGoldTAS` : Micro Gold TAS Futures (COMEX: MGT)
- `Futures.Metals.MicroPalladium` : Micro Palladium Futures (NYMEX: PAM)
- `Futures.Metals.MicroSilver` : Micro Silver Futures (COMEX: SIL)
- `Futures.Metals.Palladium` : Palladium Futures (NYMEX: PA)
- `Futures.Metals.Platinum` : Platinum Futures (NYMEX: PL)
- `Futures.Metals.Silver` : Silver Futures (COMEX: SI)
- `Futures.Metals.USMidwestDomesticHotRolledCoilSteelCRUIndex` : U.S. Midwest Domestic Hot-Rolled Coil Steel (CRU) Index Futures (NYMEX: HRC)
- `Futures.Softs.Sugar11` : Sugar #11 Futures ICE (ICE: SB)
- `Futures.Softs.Sugar11CME` : Sugar #11 Futures CME (NYMEX: YO)

Example Applications

The US Future Universe dataset enables you to accurately design Futures strategies. Examples include the following strategies:

- Buying the Futures contract with the most open interest to reduce slippage and market impact
- Trading bull calendar spreads to reduce volatility and margin requirements
- Trading bull calendar spreads to speculate contango

Classic Algorithm Example

The following example algorithm selects and buys the front-month Mini Gold Futures contract and sells the front-month Micro Gold Futures contract. When the front-month contract changes, the algorithm rebalances the portfolio.

C#

```
public class USFuturesDataAlgorithm : QCAlgorithm
{
    private Future _miniGold;
    private Future _microGold;
    // Save a cache of the mapped symbol to trade.
    private Dictionary<Symbol, FuturesContract?> _contract = new ();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);
        UniverseSettings.Asynchronous = true;

        // Requesting data on gold and micro-gold contracts.
        // Filter the universe to trade only the contracts expiring within 90 days to ensure liquidity.
        _miniGold = AddFuture(Futures.Metals.Gold);
        _miniGold.SetFilter(0, 90);
        _contract.Add(_miniGold.Symbol, null);

        _microGold = AddFuture(Futures.Metals.MicroGold);
        _microGold.SetFilter(0, 90);
        _contract.Add(_microGold.Symbol, null);
    }

    public override void OnData(Slice slice)
    {
        foreach (var (symbol, chain) in slice.FutureChains)
        {
            if (_contract.ContainsKey(symbol))
            {
                // Select the contract with the greatest open interest to trade with the most efficiency.
                var mostLiquidContract = chain.OrderBy(x => x.OpenInterest).Last();

                if (_contract[symbol] == null || mostLiquidContract.Symbol != _contract[symbol].Symbol)
                {
                    // Liquidate any unmapped contracts.
                    if (_contract[symbol] != null)
                    {
                        Liquidate(_contract[symbol].Symbol);
                    }
                    _contract[symbol] = mostLiquidContract;

                    // Buy mini-gold and short micro-gold contracts as planned.
                    if (symbol == _miniGold.Symbol)
                    {
                        MarketOrder(_contract[symbol].Symbol, 1);
                    }
                    else if (symbol == _microGold.Symbol)
                    {
                        MarketOrder(_contract[symbol].Symbol, -1);
                    }
                }
            }
        }
    }
}
```

Framework Algorithm Example

The following example algorithm selects and buys the front-month Mini Gold Futures contract and sells the front-month Micro Gold Futures contract. When the front-month contract changes, the algorithm rebalances the portfolio.

C#

```
public class USFuturesDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);
        UniverseSettings.Asynchronous = true;
        // Set up an universe selection model that selects the front month contract
        SetUniverseSelection(new FrontMonthFutureUniverseSelectionModel());
        SetAlpha(new ConstantFuturesAlphaModel());
```

```
        SetAlpha(new ConstantFuturesAlphaModel());
        // A portfolio construction model that only order a single share per insight signal
        SetPortfolioConstruction(new SingleSharePortfolioConstructionModel());
    }
}

class FrontMonthFutureUniverseSelectionModel : FutureUniverseSelectionModel
{
    // Daily updating with select_future_chain_symbols function
    public FrontMonthFutureUniverseSelectionModel()
        : base(TimeSpan.FromDays(1), SelectFutureChainSymbols) {}

    private static IEnumerable<Symbol> SelectFutureChainSymbols(DateTime utcTime)
    {
        //Select gold and micro gold contracts for the strategy need
        return new List<Symbol> {
            QuantConnect.Symbol.Create(Futures.Metals.Gold, SecurityType.Future, Market.COMEX),
            QuantConnect.Symbol.Create(Futures.Metals.MicroGold, SecurityType.Future, Market.COMEX)
        };
    }

    protected override FutureFilterUniverse Filter(FutureFilterUniverse filter)
    {
        // Filter only front month contract for liquidity and most informed information
        return filter.FrontMonth().OnlyApplyFilterAtMarketOpen();
    }
}


class ConstantFuturesAlphaModel : AlphaModel
{
    // Long gold and short micro gold in this strategy
    private Symbol
        _longSymbol = QuantConnect.Symbol.Create(Futures.Metals.Gold, SecurityType.Future, Market.COMEX),
        _shortSymbol = QuantConnect.Symbol.Create(Futures.Metals.MicroGold, SecurityType.Future, Market.COMEX);

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = new List<Insight>();

        if (algorithm.Portfolio.Invested)
        {
            return insights;
        }

        // For both gold and micro gold, select the front month contract (only contract) in the chain
        foreach (var kvp in slice.FutureChains)
        {
            var symbol = kvp.Key;
            var chain = kvp.Value;
            var contract = chain.First();

            // Long gold and short micro gold as planned
            if (symbol == _longSymbol)
            {
                insights.Add(Insight.Price(contract.Symbol, contract.Expiry + TimeSpan.FromDays(1),
InsightDirection.Up));
            }
            else if (symbol == _shortSymbol)
            {
                insights.Add(Insight.Price(contract.Symbol, contract.Expiry + TimeSpan.FromDays(1),
InsightDirection.Down));
            }
        }

        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            // Historical data
            var history = algorithm.History(security.Symbol, 100, Resolution.Minute);
            algorithm.Debug($"We got {history.Count()} from our history request for {security.Symbol}");
        }
    }
}


class SingleSharePortfolioConstructionModel : PortfolioConstructionModel
{
```

```
{
    public override IEnumerable<PortfolioTarget> CreateTargets(QCAlgorithm algorithm, Insight[] insights)
    {
        var targets = new List<PortfolioTarget>();
        foreach (var insight in insights)
        {
            if (algorithm.Securities[insight.Symbol].IsTradable)
            {
                // Single share only using integer portfolio target
                targets.Add(new PortfolioTarget(insight.Symbol, (int) insight.Direction));
            }
        }
        return targets;
    }
}
```

Data Point Attributes

The US Future Universe dataset provides `FutureFilterUniverse` and `FutureUniverse` objects.

FutureFilterUniverse Attributes

`FutureFilterUniverse` objects have the following attributes:

FutureUniverse Attributes

`FutureUniverse` objects have the following attributes:

QuantConnect

US Futures Security Master

Introduction

The US Futures Security Master dataset by QuantConnect provides mapping reference data for the most liquid contracts of the CME Group exchanges, calculated with popular rolling techniques. The data covers 157 root Future contracts, starts in May 2009, and is delivered on a daily frequency with a zip file with all the contract mappings. This dataset is created by daily processing of the US historical Future chains.

This dataset, paired with the US Futures dataset, supports the following rolling techniques: forward panama canal, backwards panama canal, and backwards ratio. You can set the specific rolling date to occur on the last trading day, first day month, or the day when the contract with the greatest open interest changes.

VIX Futures don't support continous contract rolling with open interest.

This is `NOT` the underlying Futures data (US Futures dataset), which you need to purchase separately with a license from AlgoSeek. This security master dataset is required to purchase the US Futures or US Future Options datasets.

For more information about the US Futures Security Master dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Getting Started

You don't need any special code to utilize the US Futures Security Master. It automatically loads when you request US Futures data .

Data Summary

The following table describes the dataset properties:

Property	**Value**
Start Date	May 2009
Asset Coverage	157 Liquid Futures
Data Density	Regular
Resolution	Daily

VIX Futures don't support continous contract rolling with open interest.

Key Concept

The US Futures Security Master lets you to construct continuous Futures, allowing the access of normalized historical data of the underlying assets, as well as trading the “lead” Future contracts for those assets.

Continuous Futures refer to sets of rolling lead Future contracts during their actively trading periods. Since Future contracts expire at their maturities, to analyze the historical price of a Future and to apply technical indicators, you need the continuous Future price series.

To access the continuous Future, use the Future's `Symbol` property.

```C#
// Set the data normalization mode and data mapping mode to select the contract as the continuous contract with price
adjustment for smooth price series
var future = AddFuture(Futures.Energy.CrudeOilWTI,
    dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
    dataMappingMode: DataMappingMode.OpenInterest,
    contractDepthOffset: 0
);
_continuousFutureSymbol = future.Symbol;
```

The `dataNormalizationMode` and `dataMappingMode` arguments makes the transition of the underlying contracts seemless. However, the Future `Symbol` doesn't map to an underlying Future contract. It works fine to trade within backtests, but could be subjected to friction costs during live trading since the order price could be a normalized price. For more information about this topic, see the `Live Trading Considerations` section.

Data Normalization Modes

The data normalization mode defines how the price series of two contracts are stitched together when the contract rollovers occur. The `DataNormalizatoinMode` enumeration has the following members available for continuous contracts:

If you use a data normalization mode that's not in the preceding list, LEAN automatically converts it to `DataNormalizationMode.BackwardsRatio` .

Data Mapping Modes

The data mapping mode defines when contract rollovers occur. The `DataMappingMode` enumeration has the following members:

VIX Futures don't support continous contract rolling with `DataMappingMode.OpenInterest` and `DataMappingMode.OpenInterestAnnuak` .

Tracking Contract Changes

As the contracts roll over, the `Mapped` property of the `Future` object references the next contract in the series and you receive a `SymbolChangedEvent` . To get the current `Symbol` change events, index the `SymbolChangedEvents` property of the current `Slice` with the continuous Futures `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your Future at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```C#
public override void OnData(Slice slice)
{
    // Get Symbol Change Event of the Continuous Future (change in mapped contract)
    if (slice.SymbolChangedEvents.TryGetValue(_continuousFutureSymbol, out var changedEvent))
    {
        var oldSymbol = changedEvent.OldSymbol;
        var newSymbol = AddFutureContract(changedEvent.NewSymbol).Symbol;
        var tag = $"Rollover - Symbol changed at {Time}: {oldSymbol} -> {newSymbol}";
        var quantity = Portfolio[oldSymbol].Quantity;
        // Rolling over: to liquidate any position of the old mapped contract and switch to the newly mapped contract
        Liquidate(oldSymbol, tag: tag);
        if (quantity != 0) MarketOrder(newSymbol, quantity, tag: tag);
        Log(tag);
    }
}
```

`SymbolChangedEvent` objects have the following attributes:

Live Trading Considerations

You can trade continuous Futures, but the continuous Future `Symbol` doesn't map to a single underlying Future contract. Instead, it represents a set of rolling contracts. Thus, the prices could be frictional during a contract rollover, which could be catastrophic in live trading! For live trading, you should place your orders directly on the underlying contracts. To get the current underlying contract in the continuous Future series, use the `Mapped` property.

```C#
// Get the mapped symbol
var currentContract = _continuousContract.Mapped;
Buy(currentContract, 1);
```

Data Format

If you download the files in the US Futures Security Master, you get a factor file and a map file for each of the exchanges with supported continuous Futures. To view the files, see the *|data|future|<exchange_name>* directory under your LEAN CLI base directory.

For the factor file, it is a `.zip` collection of REST API styled `.csv` files for each Future `Symbol`, including the date, scaling factors for each type of data normalization and the data mapping mode that indicates the Symbol changing event is on that day for that mapping mode. The following line is an example line in the `.csv` file:

{"Date":"2009-10-31T00:00:00","BackwardsRatioScale":[0.99141630901287553648068669 53,1.0,1.0],"BackwardsPanamaCanalScale":[-2.0,0.0,0.0],"ForwardPanamaCanalScale":[0.0,0.0,0.0],"DataMappingMode":1}

For the map file, it is a `.zip` collection of `.csv` files for each Future `Symbol`, including the date, new underlying contract Symbol, the exchange code, and the data mapping mode that indicates the Symbol changing event is on that day for that mapping mode. The following line is an example line in the `.csv` file:

```
20091130,aw uii3j0m6zbj9,CBOT,1
```

Historical Data

You can get historical US Future Security Master data in an algorithm and the Research Environment.

Historical Data In Algorithms

To get historical US Future Security Master data in an algorithm, call the `History<SymbolChangedEvent>` method with continuous Future `Symbol` and a lookback period. This method returns the old and new contract of each rollover. If there is no data for the period you requested, the history result is empty.

```
// Add the Future and define its rollover settings.
var future = AddFuture(
    Futures.Indices.SP500EMini,
    dataMappingMode: DataMappingMode.OpenInterest,
    dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
    contractDepthOffset: 0
);

// Get historical data.
var history = History<SymbolChangedEvent>(future.Symbol, TimeSpan.FromDays(365));
```

For more information about historical US Future Security Master data in algorithms, see Symbol Changes .

Historical Data In Research

To get historical US Future Security Master data in the Research Environment, call the `History<SymbolChangedEvent>` method with continuous Future `Symbol` and a time period. This method returns the old and new contract of each rollover. If there is no data for the period you requested, the history result is empty.

```
// Add the Future and define its rollover settings.
var future = qb.AddFuture(
    Futures.Indices.SP500EMini,
    dataMappingMode: DataMappingMode.OpenInterest,
    dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
    contractDepthOffset: 0
);

// Get historical data.
var history = qb.History<SymbolChangedEvent>(future.Symbol, new DateTime(2023, 1, 1), new DateTime(2025, 1, 1));
```

For more information about historical US Future Security Master data in the Research Environment, see Symbol Changes History .

Supported Assets

The following list shows the available (157) Futures:

- `Futures.Currencies.AUD` : Australian Dollar Futures (CME: 6A)
- `Futures.Currencies.AUDCAD` : Australian Dollar/Canadian Dollar Futures (CME: ACD)
- `Futures.Currencies.AUDJPY` : Australian Dollar/Japanese Yen Futures (CME: AJY)
- `Futures.Currencies.AUDNZD` : Australian Dollar/New Zealand Dollar Futures (CME: ANE)
- `Futures.Currencies.BRL` : Brazillian Real Futures (CME: 6L)
- `Futures.Currencies.BTC` : Bitcoin Futures (CME: BTC)
- `Futures.Currencies.BTICMicroBTC` : BTIC on Micro Bitcoin Futures (CME: MIB)
- `Futures.Currencies.BTICMicroEther` : BTIC on Micro Ether Futures (CME: MRB)
- `Futures.Currencies.CAD` : Canadian Dollar Futures (CME: 6C)
- `Futures.Currencies.CADJPY` : Canadian Dollar/Japanese Yen Futures (CME: CJY)
- `Futures.Currencies.CHF` : Swiss Franc Futures (CME: 6S)
- `Futures.Currencies.ETH` : Ether Futures (CME: ETH)
- `Futures.Currencies.EUR` : Euro FX Futures (CME: 6E)
- `Futures.Currencies.EURAUD` : Euro/Australian Dollar Futures (CME: EAD)
- `Futures.Currencies.EURCAD` : Euro/Canadian Dollar Futures (CME: ECD)
- `Futures.Currencies.EuroFXEmini` : E-mini Euro FX Futures (CME: E7)
- `Futures.Currencies.EURSEK` : Euro/Swedish Krona Futures (CME: ESK)

- `Futures.Currencies.GBP` : British Pound Futures (CME: 6B)
- `Futures.Currencies.JapaneseYenEmini` : E-mini Japanese Yen Futures (CME: J7)
- `Futures.Currencies.JPY` : Japanese Yen Futures (CME: 6J)
- `Futures.Currencies.MicroAUD` : Micro AUD/USD Futures (CME: M6A)
- `Futures.Currencies.MicroBTC` : Micro Bitcoin Futures (CME: MBT)
- `Futures.Currencies.MicroCAD` : Micro USD/CAD Futures (CME: M6C)
- `Futures.Currencies.MicroCADUSD` : Micro CAD/USD Futures (CME: MCD)
- `Futures.Currencies.MicroCHF` : Micro CHF/USD Futures (CME: MSF)
- `Futures.Currencies.MicroEther` : Micro Ether Futures (CME: MET)
- `Futures.Currencies.MicroEUR` : Micro EUR/USD Futures (CME: M6E)
- `Futures.Currencies.MicroGBP` : Micro GBP/USD Futures (CME: M6B)
- `Futures.Currencies.MicroINRUSD` : Micro INR/USD Futures (CME: MIR)
- `Futures.Currencies.MicroJPY` : Micro JPY/USD Futures (CME: MJY)
- `Futures.Currencies.MicroUSDCHF` : Micro USD/CHF Futures (CME: M6S)
- `Futures.Currencies.MicroUSDCNH` : Micro USD/CNH Futures (CME: MNH)
- `Futures.Currencies.MicroUSDJPY` : Micro USD/JPY Futures (CME: M6J)
- `Futures.Currencies.MXN` : Mexican Peso Futures (CME: 6M)
- `Futures.Currencies.NZD` : New Zealand Dollar Futures (CME: 6N)
- `Futures.Currencies.RUB` : Russian Ruble Futures (CME: 6R)
- `Futures.Currencies.StandardSizeUSDOffshoreRMBCNH` : Standard-Size USD/Offshore RMB (CNH) Futures (CME: CNH)
- `Futures.Currencies.ZAR` : South African Rand Futures (CME: 6Z)
- `Futures.Energy.ArgusLLSvsWTIArgusTradeMonth` : Argus LLS vs. WTI (Argus) Trade Month Futures (NYMEX: AE5)
- `Futures.Energy.ArgusPropaneFarEastIndex` : Argus Propane Far East Index Futures (NYMEX: A7E)
- `Futures.Energy.ArgusPropaneSaudiAramco` : Argus Propane (Saudi Aramco) Futures (NYMEX: A9N)
- `Futures.Energy.BrentCrudeOilVsDubaiCrudeOilPlatts` : Brent Crude Oil vs. Dubai Crude Oil (Platts) Futures (NYMEX: ADB)
- `Futures.Energy.BrentLastDayFinancial` : Brent Last Day Financial Futures (NYMEX: BZ)
- `Futures.Energy.ChicagoEthanolPlatts` : Chicago Ethaanol (Platts) Futures (NYMEX: CU)
- `Futures.Energy.ConwayPropaneOPIS` : Conway Propane (OPIS) Futures (NYMEX: A8K)
- `Futures.Energy.CrudeOilWTI` : Crude Oil WTI Futures (NYMEX: CL)
- `Futures.Energy.DubaiCrudeOilPlattsFinancial` : Dubai Crude Oil (Platts) Financial Futures (NYMEX: DCB)
- `Futures.Energy.EastWestGasolineSpreadPlattsArgus` : East-West Gasoline Spread (Platts-Argus) Futures (NYMEX: EWG)
- `Futures.Energy.EastWestNaphthaJapanCFvsCargoesCIFNWESpreadPlatts` : East-West Naphtha: Japan C&F vs. Cargoes CIF NWE Spread (Platts) Futures (NYMEX: EWN)
- `Futures.Energy.Ethanol` : Ethanol Futures (CBOT: EH)
- `Futures.Energy.EthanolT2FOBRdamIncludingDutyPlatts` : Ethanol T2 FOB Rdam Including Duty (Platts) Futures (NYMEX: AZ1)
- `Futures.Energy.EuropeanNaphthaPlattsCrackSpread` : European Naphtha (Platts) Crack Spread Futures (NYMEX: EN)
- `Futures.Energy.EuropeanPropaneCIFARAArgus` : European Propane CIF ARA (Argus) Futures (NYMEX: APS)
- `Futures.Energy.EuropeanPropaneCIFARAArgusVsNaphthaCargoesCIFNWEPlatts` : European Propane CIF ARA (Argus) vs. Naphtha Cargoes CIF NWE (Platts) Futures (NYMEX: EPN)
- `Futures.Energy.FreightRouteTC14Baltic` : Freight Route TC14 (Baltic) Futures (NYMEX: FRC)
- `Futures.Energy.Gasoline` : Gasoline RBOB Futures (NYMEX: RB)
- `Futures.Energy.GasolineEurobobOxyNWEBargesArgus` : Gasoline Euro-bob Oxy NWE Barges (Argus) Futures (NYMEX:

B7H)

- `Futures.Energy.GroupThreeSuboctaneGasolinePlattsVsRBOB` : Group Three Sub-octane Gasoliine (Platts) vs. RBOB Futures (NYMEX: AA8)
- `Futures.Energy.GroupThreeULSDPlattsVsNYHarborULSD` : Group Three ULSD (Platts) vs. NY Harbor ULSD Futures (NYMEX: AA6)
- `Futures.Energy.GulfCoastCBOBGasolineA2PlattsVsRBOBGasoline` : Gulf Coast CBOB Gasoline A2 (Platts) vs. RBOB Gasoline Futures (NYMEX: CRB)
- `Futures.Energy.GulfCoastHSFOPlattsVsEuropeanThreePointFivePercentFuelOilBargesFOBRdamPlatts` : Gulf Coast HSFO (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures (NYMEX: GCU)
- `Futures.Energy.HeatingOil` : Heating Oil Futures (NYMEX: HO)
- `Futures.Energy.LosAngelesCARBOBGasolineOPISvsRBOBGasoline` : Los Angeles CARBOB Gasoline (OPIS) vs. RBOB Gasoline Futures (NYMEX: AJL)
- `Futures.Energy.LosAngelesCARBDieselOPISvsNYHarborULSD` : Los Angeles CARB Diesel (OPIS) vs. NY Harbor ULSD Futures (NYMEX: AKL)
- `Futures.Energy.LosAngelesJetOPISvsNYHarborULSD` : Los Angeles Jet (OPIS) vs. NY Harbor ULSD Futures (NYMEX: AJS)
- `Futures.Energy.MarsArgusVsWTIFinancial` : Mars (Argus) vs. WTI Financial Futures (NYMEX: AYX)
- `Futures.Energy.MarsArgusVsWTITradeMonth` : Mars (Argus) vs. WTI Trade Month Futures (NYMEX: AYV)
- `Futures.Energy.MicroCrudeOilWTI` : Micro WTI Crude Oil Futures (NYMEX: MCL)
- `Futures.Energy.MicroEuropeanFOBRdamMarineFuelZeroPointFivePercentBargesPlatts` : Micro European FOB Rdam Marine Fuel 0.5% Barges (Platts) Futures (NYMEX: R5O)
- `Futures.Energy.MicroEuropeanThreePointFivePercentOilBargesFOBRdamPlatts` : Micro European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures (NYMEX: MEF)
- `Futures.Energy.MicroGasoilZeroPointOnePercentBargesFOBARAPlatts` : Micro Gasoil 0.1% Barges FOB ARA (Platts) Futures (NYMEX: M1B)
- `Futures.Energy.MicroSingaporeFOBMarineFuelZeroPointFivePercetPlatts` : Micro Singapore FOB Marine Fuel 0.5% (Platts) Futures (NYMEX: S5O)
- `Futures.Energy.MicroSingaporeFuelOil380CSTPlatts` : Micro Singapore Fuel Oil 380CST (Platts) Futures (NYMEX: MAF)
- `Futures.Energy.MiniEuropeanThreePointPercentFiveFuelOilBargesPlatts` : Mini European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures (NYMEX: A0D)
- `Futures.Energy.MiniSingaporeFuelOil180CstPlatts` : Mini Singapore Fuel Oil 180 cst (Platts) Futures (NYMEX: A0F)
- `Futures.Energy.MontBelvieuEthaneOPIS` : Mont Belvieu Ethane (OPIS) Futures (NYMEX: AC0)
- `Futures.Energy.MontBelvieuLDHPropaneOPIS` : Mont Belvieu LDH Propane (OPIS) Futures (NYMEX: B0)
- `Futures.Energy.MontBelvieuNaturalGasolineOPIS` : Mont Belvieu Natural Gasoline (OPIS) Futures (NYMEX: A7Q)
- `Futures.Energy.MontBelvieuNormalButaneOPIS` : Mont Belvieu Normal Butane (OPIS) Futures (NYMEX: AD0)
- `Futures.Energy.NaturalGas` : Natural Gas Futures (NYMEX: NG)
- `Futures.Energy.NaturalGasHenryHubLastDayFinancial` : Natural Gas (Henry Hub) Last-day Financial Futures (NYMEX: HH)
- `Futures.Energy.NaturalGasHenryHubPenultimateFinancial` : Natural Gas (Henry Hub) Penultimate Financial Futures (NYMEX: HP)
- `Futures.Energy.OnePercentFuelOilCargoesFOBNWEPlattsVsThreePointFivePercentFuelOilBargesFOBRdamPlatts` : 1% Fuel Oil Cargoes FOB NWE (Platts) vs. 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures (NYMEX: FSS)
- `Futures.Energy.PremiumUnleadedGasoline10ppmFOBMEDPlatts` : Premium Unleaded Gasoline 10 ppm FOB MED (Platts) Futures (NYMEX: A3G)
- `Futures.Energy.PropaneNonLDHMontBelvieuOPIS` : Propane Non-LDH Mont Belvieu (OPIS) Futures (NYMEX: A1R)
- `Futures.Energy.RBOBGasolineCrackSpread` : RBOB Gasoline Crack Spread Futures (NYMEX: ARE)

- `Futures.Energy.RBOBGasolineVsEurobobOxyNWEBargesArgusThreeHundredFiftyThousandGallons` : RBOB Gasoline vs. Euro-bob Oxy NWE Barges (Argus) (350,000 gallons) Futures (NYMEX: EXR)
- `Futures.Energy.SingaporeFuelOil380cstPlattsVsEuropeanThreePointFivePercentFuelOilBargesFOBRdamPlatts` : Singapore Fuel Oil 380 cst (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures (NYMEX: EVC)
- `Futures.Energy.SingaporeGasoilPlattsVsLowSulphurGasoilFutures` : Singapore Gasoil (Platts) vs. Low Sulphur Gasoil Futures (NYMEX: AGA)
- `Futures.Energy.SingaporeMogas92UnleadedPlattsBrentCrackSpread` : Singapore Mogas 92 Unleaded (Platts) Brent Crack Spread Futures (NYMEX: D1N)
- `Futures.Energy.ThreePointFivePercentFuelOilBargesFOBRdamPlattsCrackSpread` : 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread Futures (NYMEX: FO)
- `Futures.Energy.ThreePointFivePercentFuelOilBargesFOBRdamPlattsCrackSpread1000mt` : 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread (1000mt) Futures (NYMEX: BOO)
- `Futures.Energy.WTIBrentFinancial` : WTI-Brent Financial Futures (NYMEX: BK)
- `Futures.Energy.WTIFinancial` : WTI Financial Futures (NYMEX: CSX)
- `Futures.Energy.WTIHoustonArgusVsWTITradeMonth` : WTI Houston (Argus) vs. WTI Trade Month Futures (NYMEX: HTT)
- `Futures.Energy.WTIHoustonCrudeOil` : WTI Houston Crude Oil Futures (NYMEX: HCL)
- `Futures.Financials.EuroDollar` : EuroDollar Futures (CME: GE)
- `Futures.Financials.FiveYearUSDMACSwap` : 5-Year USD MAC Swap Futures (CBOT: F1U)
- `Futures.Financials.MicroY10TreasuryNote` : Micro 10-Year Yield Futures (CBOT: 10Y)
- `Futures.Financials.MicroY2TreasuryBond` : Micro 2-Year Yield Futures (CBOT: 2YY)
- `Futures.Financials.MicroY30TreasuryBond` : Micro 30-Year Yield Futures (CBOT: 30Y)
- `Futures.Financials.MicroY5TreasuryBond` : Micro 5-Year Yield Futures (CBOT: 5YY)
- `Futures.Financials.UltraTenYearUSTreasuryNote` : Ultra 10-Year U.S. Treasury Note Futures (CBOT: TN)
- `Futures.Financials.UltraUSTreasuryBond` : Ultra U.S. Treasury Bond Futures (CBOT: UB)
- `Futures.Financials.Y10TreasuryNote` : 10Y U.S. Treasury Note Futures (CBOT: ZN)
- `Futures.Financials.Y2TreasuryNote` : 2Y U.S. Treasury Note Futures (CBOT: ZT)
- `Futures.Financials.Y30TreasuryBond` : 30Y U.S. Treasury Bond Futures (CBOT: ZB)
- `Futures.Financials.Y5TreasuryNote` : 5Y U.S. Treasury Note Futures (CBOT: ZF)
- `Futures.Forestry.Lumber` : Lumber Futures (CME: LBR)
- `Futures.Forestry.RandomLengthLumber` : Random Length Lumber Futures (CME: LBS)
- `Futures.Grains.BlackSeaCornFinanciallySettledPlatts` : Black Sea Corn Financially Settled (Platts) Futures (CBOT: BCF)
- `Futures.Grains.BlackSeaWheatFinanciallySettledPlatts` : Black Sea Wheat Financially Settled (Platts) Futures (CBOT: BWF)
- `Futures.Grains.Corn` : Corn Futures (CBOT: ZC)
- `Futures.Grains.HRWWheat` : KC HRW Wheat Futures (CBOT: KE)
- `Futures.Grains.Oats` : Oats Futures (CBOT: ZO)
- `Futures.Grains.Soybeans` : Soybeans Futures (CBOT: ZS)
- `Futures.Grains.SoybeanMeal` : Soybean Meal Futures (CBOT: ZM)
- `Futures.Grains.SoybeanOil` : Soybean Oil Futures (CBOT: ZL)
- `Futures.Grains.Wheat` : Default wheat contract is SRWWheat (CBOT: ZW)
- `Futures.Indices.BloombergCommodityIndex` : Bloomberg Commodity Index Futures (CBOT: AW)
- `Futures.Indices.Dow30EMini` : E-mini Dow Indu 30 Futures (CBOT: YM)
- `Futures.Indices.DowJonesRealEstate` : Dow Jones Real Estate futures on CME (CME: RX)
- `Futures.Indices.FTSEEmergingEmini` : E-mini FTSE Emerging Index Futures (CME: EI)

- `Futures.Indices.MicroDow30EMini` : Micro E-mini Dow Jones Industrial Average Index Futures (CBOT: MYM)
- `Futures.Indices.MicroNASDAQ100EMini` : Micro E-mini Nasdaq-100 Index Futures (CME: MNQ)
- `Futures.Indices.MicroRussell2000EMini` : Micro E-mini Russell 2000 Index Futures (CME: M2K)
- `Futures.Indices.MicroSP500EMini` : Micro E-mini S&P 500 Index Futures (CME: MES)
- `Futures.Indices.NASDAQ100BiotechnologyEMini` : E-mini Nasdaq-100 Biotechnology Index Futures (CME: BIO)
- `Futures.Indices.NASDAQ100EMini` : E-mini NASDAQ 100 Futures (CME: NQ)
- `Futures.Indices.Nikkei225Dollar` : Nikkei-225 Dollar Futures (CME: NKD)
- `Futures.Indices.Nikkei225YenCME` : Nikkei-225 Yen denominated Futures on CME (CME: NIY)
- `Futures.Indices.Russell1000EMini` : E-mini Russell 1000 futures on CME (CME: RS1)
- `Futures.Indices.Russell2000EMini` : E-mini Russell 2000 Futures (CME: RTY)
- `Futures.Indices.SPGSCICommodity` : S&P-GSCI Commodity Index Futures (CME: GD)
- `Futures.Indices.SP400MidCapEmini` : E-mini S&P MidCap 400 Futures (CME: EMD)
- `Futures.Indices.SP500AnnualDividendIndex` : (CME: SDA)
- `Futures.Indices.SP500EMini` : E-mini S&P 500 Futures (CME: ES)
- `Futures.Indices.TOPIXYEN` : YEN Denominated Topix Index Futures on CME (CME: TPY)
- `Futures.Indices.USDDenominatedIbovespa` : USD-Denominated Ibovespa Index Futures (CME: IBV)
- `Futures.Indices.VIX` : CBOE Volatility Index Futures (CFE: VX)
- `Futures.Meats.FeederCattle` : Feeder Cattle Futures (CME: GF)
- `Futures.Meats.LeanHogs` : Lean Hogs Futures (CME: HE)
- `Futures.Meats.LiveCattle` : Live Cattle Futures (CME: LE)
- `Futures.Metals.AluminiumEuropeanPremiumDutyPaidMetalBulletin` : Aluminium European Premium Duty-Paid (Metal Bulletin) Futures (COMEX: EDP)
- `Futures.Metals.AluminumMWUSTransactionPremiumPlatts25MT` : Aluminum MW U.S. Transaction Premium Platts (25MT) Futures (COMEX: AUP)
- `Futures.Metals.Copper` : Copper Futures (COMEX: HG)
- `Futures.Metals.Gold` : Gold Futures (COMEX: GC)
- `Futures.Metals.MicroGold` : Micro Gold Futures (COMEX: MGC)
- `Futures.Metals.MicroGoldTAS` : Micro Gold TAS Futures (COMEX: MGT)
- `Futures.Metals.MicroPalladium` : Micro Palladium Futures (NYMEX: PAM)
- `Futures.Metals.MicroSilver` : Micro Silver Futures (COMEX: SIL)
- `Futures.Metals.Palladium` : Palladium Futures (NYMEX: PA)
- `Futures.Metals.Platinum` : Platinum Futures (NYMEX: PL)
- `Futures.Metals.Silver` : Silver Futures (COMEX: SI)
- `Futures.Metals.USMidwestDomesticHotRolledCoilSteelCRUIndex` : U.S. Midwest Domestic Hot-Rolled Coil Steel (CRU) Index Futures (NYMEX: HRC)
- `Futures.Softs.Sugar11` : Sugar #11 Futures ICE (ICE: SB)
- `Futures.Softs.Sugar11CME` : Sugar #11 Futures CME (NYMEX: YO)

Example Applications

The US Futures Security Master enables you to design strategies harnessing continuous Futures contracts. Examples include the following strategies:

- Trading cyclical patterns in commodity Futures.
- Buying gold Futures as an inflation hedge with automatic contract roll-overs.
- Detecting arbitrage opportunities between index Futures and Equities.

Classic Algorithm Example

The following example algorithm buys the continuous Future contract for Crude Oil when its price rises above its simple moving average. When its price drops below its simple moving average, the algorithm sells the continuous contract. Each time the continuous Future contract rolls over, the algorithm logs the event.

C#

```
public class USFuturesSecurityMasterDataClassicAlgorithm  : QCAlgorithm
{
    private Future _future;
    // Define a threshold for the SMA cross.
    private decimal _threshold = 0.01m;
    private SimpleMovingAverage _sma;

    public override void Initialize()
    {
        SetCash(1000000);
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Seed the price of each asset with its last known price to
        // avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Add a Future. Set the continuous contract mapping criteria to
        // the contract with highest open interest to provide the best
        // price information for trend estimation.
        _future = AddFuture(
            Futures.Energy.CrudeOilWTI,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            dataMappingMode: DataMappingMode.OpenInterest,
            contractDepthOffset: 0
        );
        // Historical data: Get the contract rollovers for the trailing
        // year.
        var history = History<SymbolChangedEvent>(_future.Symbol, TimeSpan.FromDays(365));
        foreach (var symbolChangedEvent in history)
        {
            var t = symbolChangedEvent.EndTime;
            var oldSymbol = symbolChangedEvent.OldSymbol;
            var newSymbol = symbolChangedEvent.NewSymbol;
        }
        // Set up an SMA indicator for to estimate the trend direction.
        _sma = SMA(_future.Symbol, 10, Resolution.Daily);
        // Warm up the SMA indicator so we can trade right away.
        WarmUpIndicator(_future.Symbol, _sma);
    }

    public override void OnData(Slice slice)
    {
        // Wait until we have the latest TradeBar and the indicator is
        // ready.
        if (!(slice.Bars.ContainsKey(_future.Symbol) && _sma.IsReady))
        {
            return;
        }
        var contract = Portfolio[_future.Mapped];
        int quantity;
        // Long during an uptrend.
        if (!contract.IsLong && _future.Price > _sma.Current.Value * (1+_threshold))
        {
            quantity = 1;
        }
        // Short during an downtrend.
        else if (!contract.IsShort && _future.Price < _sma.Current.Value * (1-_threshold))
        {
            quantity = -1;
        }
        else
        {
            return;
        }
        MarketOrder(_future.Mapped, quantity - contract.Quantity);
    }

    public override void OnSymbolChangedEvents(SymbolChangedEvents symbolChangedEvents)
    {
        foreach (var (symbol, changedEvent) in symbolChangedEvents)
        {
            var oldSymbol = changedEvent.OldSymbol;
```

```
            var newSymbol = changedEvent.NewSymbol;
            // The quantity to roll over should be consistent.
            var quantity = Portfolio[oldSymbol].Quantity;
            if (quantity == 0)
            {
                continue;
            }
            // Rolling over: Liquidate the old mapped contract and switch
            // to the new mapped contract.
            var tag = $"Rollover: {oldSymbol} -> {newSymbol}";
            Liquidate(oldSymbol, tag: tag);
            MarketOrder(newSymbol, quantity, tag: tag);
        }
    }
}
```

Framework Algorithm Example

The following example algorithm buys the continuous Future contract for Crude Oil when its price rises above its simple moving average. When its price drops below its simple moving average, the algorithm sells the continuous contract. Each time the contract rolls over, the algorithm logs the event.

C#

```
public class USFuturesSecurityMasterDataFrameworkAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetCash(1000000);
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Setting the continuous contract mapping criteria, the contract with highest open interest provide the best
price information for trend estimation
        AddFuture(Futures.Energies.CrudeOilWTI,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            dataMappingMode: DataMappingMode.OpenInterest,
            contractDepthOffset: 0
        );
        AddAlpha(new ContinuousFuturesAlphaModel());
        SetPortfolioConstruction(new SingleSharePortfolioConstructionModel());
    }

    class ContinuousFuturesAlphaModel : AlphaModel
    {
        private Future _continuousContract;
        private Symbol _symbol;
        // 1% margin to reassure trend direction
        private decimal _threshold = 0.01m;
        private SimpleMovingAverage _sma;

        public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
        {
            var insights = new List<Insight>();

            //  Check if contract data available, since all trades are based on that only
            if (_symbol == null || _continuousContract == null)
            {
                return insights;
            }

            // Up-to-date handling of switching the mapped contract for trade liquidity
            foreach (var (symbol, changedEvent) in slice.SymbolChangedEvents)
            {
                var oldSymbol = changedEvent.OldSymbol;
                if (algorithm.Insights.HasActiveInsights(oldSymbol, algorithm.UtcTime))
                {
                    var newSymbol = changedEvent.NewSymbol;
                    var tag = $"Rollover - Symbol changed at {algorithm.Time}: {oldSymbol} -> {newSymbol}";
                    var lastInsight = algorithm.Insights[oldSymbol].OrderBy(x => x.CloseTimeUtc).LastOrDefault();
                    insights.Add(Insight.Price(newSymbol, lastInsight.CloseTimeUtc, lastInsight.Direction, tag:
tag));
                    algorithm.Insights.Clear(new Symbol[] { oldSymbol });
                    algorithm.Log(tag);
                }
            }
```

```
            var mappedSymbol = _continuousContract.Mapped;
            // Make sure trade decisions are based on newly received data
            if (!slice.Bars.ContainsKey(_symbol) || !_sma.IsReady || mappedSymbol == null)
            {
                return insights;
            }

            // Long if trend up by threshold to follow the trend
            if (slice.Bars[_symbol].Price > _sma.Current.Value * (1+_threshold) &&
!algorithm.Portfolio[mappedSymbol].IsLong)
            {
                insights.Add(Insight.Price(mappedSymbol, TimeSpan.FromDays(14), InsightDirection.Up));
            }
            // Short if trend down by threshold to follow the trend
            else if (slice.Bars[_symbol].Price < _sma.Current.Value * (1-_threshold) &&
!algorithm.Portfolio[mappedSymbol].IsShort)
            {
                insights.Add(Insight.Price(mappedSymbol, TimeSpan.FromDays(14), InsightDirection.Down));
            }

            return insights;
        }

        public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
        {
            foreach (var security in changes.AddedSecurities)
            {
                var symbol = security.Symbol;
                // Set up SMA indicator for trend direction estimator, only for the canonical symbol
                if (symbol.IsCanonical())
                {
                    _continuousContract = (Future)security;
                    _symbol = symbol;
                    _sma = algorithm.SMA(_symbol, 10, Resolution.Daily);

                    // Historical data
                    var history = algorithm.History(symbol, 60*24*10, Resolution.Minute);
                    algorithm.Debug($"We got {history.Count()} from our history request for {symbol}");

                    // Warm up the SMA indicator for its readiness for immediate use
                    foreach (var bar in history)
                    {
                        _sma.Update(new IndicatorDataPoint(bar.Time, bar.Close));
                    }
                }
            }
        }
    }

    class SingleSharePortfolioConstructionModel : PortfolioConstructionModel
    {
        public override IEnumerable<PortfolioTarget> CreateTargets(QCAlgorithm algorithm, Insight[] insights)
        {
            var targets = new List<PortfolioTarget>();
            foreach (var insight in insights)
            {
                if (algorithm.Securities[insight.Symbol].IsTradable)
                {
                    targets.Add(new PortfolioTarget(insight.Symbol, (int) insight.Direction));
                }
            }
            return targets;
        }
    }
}
```

Data Point Attributes

The US Futures Security Master dataset provides `SymbolChangedEvent` objects, which have the following attributes:

QuantConnect

US Index Option Universe

Introduction

The US Index Option Universe dataset by QuantConnect lists the available US Index Options contracts and the current Implied Volatility and Greeks. The data covers European Option contracts for 3 US Indices: SPX, VIX, and NDX. It starts in January 2012 and is delivered on a daily update frequency. To create this dataset, we use our implementation of the forward tree pricing model, which accounts for the interest rate, dividend payments, and daily closing prices. The values in this dataset are the same values you can get from daily indicators with mirror Options.

This dataset `does not` contain market data. For market data, see US Index Options by AlgoSeek .

For more information about the US Index Option Universe dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

QuantConnect was founded in 2012 to serve quants everywhere with the best possible algorithmic trading technology. Seeking to disrupt a notoriously closed-source industry, QuantConnect takes a radically open-source approach to algorithmic trading. Through the QuantConnect web platform, more than 50,000 quants are served every month.

Getting Started

The following snippet demonstrates how to request data from the US Index Options Universe dataset:

```C#
var option = AddIndexOption("VIX");
option.SetFilter(universe => universe.delta(0.4m, 0.6m));
_optionSymbol = option.Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2012
Asset Coverage	7 Index Options
Data Density	Dense
Resolution	Daily
Timezone	New York

Specification Over Time

According to the SPX Options contract specification , some SPX contracts expire every month and SPXW contracts expires every day. Before 2021, you could only trade SPX contracts with the following expiration dates:

- Expires within the next 4 months
- Expires in September within the next 14 months
- Expires in January, March, or June within the next 2 years
- Expires in December within the next 3 years

During this time, SPXW didn't have 0DTE every day.

`Sources` :

- Cboe Options Exchange to List Three Long-Dated SPX Options Expirations, Beginning November 1, 2021

- S&P 500 Weekly Options Now Expire Five Days a Week

Requesting Data

To add US Index Options Universe data to your algorithm, call the `AddIndexOption` method. Save a reference to the Index Option `Symbol` so you can access the data later in your algorithm. To define which contracts should be in your universe, call the `SetFilter` method of the `IndexOption` object.

The `AddIndexOption` method provides a daily stream of Option chain data. To get the most recent daily chain, call the `OptionChain` method with the canonical Index Option Symbol. The `OptionChain` method returns data on all the tradable contracts, not just the contracts that pass your universe filter.

C#

```
public class IndexOptionsDataAlgorithm : QCAlgorithm
{
    private Symbol _indexSymbol, _standardOptionSymbol, _weeklyOptionSymbol;

    public override void Initialize()
    {
        SetStartDate(2021, 1, 1);
        SetEndDate(2021, 6, 1);
        SetCash(100000);
        UniverseSettings.Asynchronous = true;
        _indexSymbol = AddIndex("VIX").Symbol;

        var standardOption = AddIndexOption(_indexSymbol);
        standardOption.SetFilter(OptionFilter);
        _standardOptionSymbol = standardOption.Symbol;
        var standardChain = OptionChain(_standardOptionSymbol);

        var weeklyOption = AddIndexOption(_indexSymbol, "VIXW");
        weeklyOption.SetFilter(OptionFilter);
        _weeklyOptionSymbol = weeklyOption.Symbol;
        var weeklyChain = OptionChain(_weeklyOptionSymbol)
            .Where(contract => !IndexOptionSymbol.IsStandard(contract.Symbol));
    }

    private OptionFilterUniverse OptionFilter(OptionFilterUniverse universe)
    {
        // Contracts can be filtered by greeks, implied volatility, open interest:
        return universe
            .Delta(0.5m, 1.5m)
            .Gamma(0.0001m, 0.0006m)
            .Vega(0.01m, 1.5m)
            .Theta(-2.0m, -0.5m)
            .Rho(0.5m, 3.0m)
            .ImpliedVolatility(1.0m, 3.0m)
            .OpenInterest(100, 500);
    }
}
```

The Index resolution must be less than or equal to the Index Option resolution. For example, if you set the Index resolution to minute, then you must set the Index Option resolution to minute, hour, or daily.

For more information about creating US Index Option subscriptions, see Index Options .

Accessing Data

For information about accessing US Equity Options data, see Handling Data .

Historical Data

You can get historical US Index Options data in an algorithm and the Research Environment.

Historical Data In Algorithms

To get historical US Index Options Universe data in an algorithm, call the `History<OptionUniverse>` method with the canonical Index Option `Symbol` . This method returns data on all of the tradable contracts, not just the contracts that pass your universe filter. If there is no data in the period you request, the history result is empty.

```C#
// OptionUniverse objects
var history = History<OptionUniverse>(_standardOptionSymbol, TimeSpan.FromDays(3)).ToList();
```

For more information about Index Options Universe data in algorithms, see Historical Data .

Historical Data In Research

To get historical US Index Options Universe data in the Research Environment, call the `History<OptionUniverse>` method with the canonical Option `Symbol` . This method returns data on all of the tradable contracts, not just the contracts that pass your universe filter.

```C#
var qb = new QuantBook();
var indexSymbol = qb.AddIndex("VIX").Symbol;
var option = qb.AddIndexOption(indexSymbol); // or qb.AddIndexOption(indexSymbol, "VIXW");
var history = qb.History<OptionUniverse>(option.Symbol, new DateTime(2020, 6, 1), new DateTime(2020, 6, 6));
foreach (var chain in history)
{
    var endTime = chain.EndTime;
    var filteredContracts = chain.Data
        .Select(contract => contract as OptionUniverse)
        .Where(contract => contract.Greeks.Delta > 0.3m);
    foreach (var contract in filteredContracts)
    {
        var price = contract.Price;
        var iv = contract.ImpliedVolatility;
    }
}
```

For more information about historical Index Options Universe data in the Research Environment, see Universes .

Supported Assets

The following table shows the available Index Options:

Underlying Index	Underlying Ticker	Target Ticker	Standard Contracts	Weekly Contracts	Tradable on Expiry Day
NASDAQ-100	NDX		✓		
NASDAQ-100	NDX	NDXP	✓		✓
NASDAQ-100	NDX	NQX	✓	✓	✓
Russell 2000	RUT		✓		
Russell 2000	RUT	RUTW		✓	✓
S&P500	SPX		✓		
S&P500	SPX	SPXW		✓	✓
S&P500	VIX		✓		
S&P500	VIX	VIXW		✓	

For more information about each underlying Index, see Supported Indices .

Example Applications

The US Index Options Universe dataset enables you to accurately design strategies for Index Options. Examples include the following strategies:

- Buying VIX call Options to hedge against upcoming volatility
- Buying VIX put Options to capture the natural downward price movement in the VIX index
- Buying SPX put Options to protect against downward price movement in the S&P 500

Classic Algorithm Example

The following example algorithm subscribes to SPX put options that fall within delta range between -1 and -0.95, open interest range between 10 and 1000, and expire within seven days. Within this Option chain, the algorithm holds the put Option contract that has the minimum delta (closest to -1) during market hour to hedge the underlying intra-day movement completely. It avoid volatility from sentiment and only earns from inter-day movement from longer-term factors. When the contract expires, the algorithm rolls over to the next contract that meets this criteria.

```
from AlgorithmImports import * class IndexOptionsUniverseAlgorithm(QCAlgorithm): def initialize(self) -&GT; None: self.set_start_date(2024, 9, 1) self.set_end_date(2024, 12, 31) self.set_cash(10000000) # Asynchronous can use computational resources efficiently self.universe_settings.asynchronous = True # Subscribe to the underlying for the underlying position # Set the data normalization mode to raw for strike price comparability self.index = self.add_index("SPX").symbol # Requesting option data and filter for the hedge candidates option = self.add_index_option(self.index) option.set_filter(self.option_filter) self.option_symbol = option.symbol # Set scheduled event to buy a hedge option contract at market open to eliminate the intra-day movement self.schedule.on( self.date_rules.every_day(self.index), self.time_rules.after_market_open(self.index, 1), self.buy_hedge_contract ) # Set a scheduled event to sell the hedge contract before market close, since we want to earn from inter-day movement # Leave 2 minutes contingency to fill self.schedule.on( self.date_rules.every_day(self.index), self.time_rules.before_market_close(self.index, 2), self.sell_hedge_contract ) self.hedge = None def option_filter(self, universe: OptionFilterUniverse) -&GT; OptionFilterUniverse: # Select the contracts with delta very close to -1 and high open interest # This can effectively hedge most of the price change of the underlying and ensure the liquidity # Make sure the contract is expiring close for its tradbility return universe.include_weeklys().puts_only().expiration(2, 7).delta(-1, -0.95).open_interest(10, 1000) def buy_hedge_contract(self) -&GT; None: chain = self.current_slice.option_chains.get(self.option_symbol) if chain: # Order the underlying if not hold, the order size should match the option contract # Order only if option chain data ready for hedging if not self.portfolio[self.index].invested: self.market_order(self.index, self.securities[self.option_symbol].symbol_properties.contract_multiplier) # Get the contract with delta closest to -1 (lowest possible delta) contract = sorted(chain, key=lambda x: x.greeks.delta)[0] self.hedge = contract.symbol # Buy 1 deep ITM put with delta close to -1 to eliminate the intraday movement self.market_order(self.hedge, 1) def sell_hedge_contract(self) -&GT; None: # Check if any hedge contract position, if so, liquidate before market close to expose to underlying overnight movement if self.hedge: self.liquidate(self.hedge) self.hedge = None
```

```
public class IndexOptionsUniverseAlgorithm : QCAlgorithm { private Symbol _index, _optionSymbol, _hedge; public override void Initialize() { SetStartDate(2024, 9, 1); SetEndDate(2024, 12, 31); SetCash(10000000); // Asynchronous can use computational resources efficiently UniverseSettings.Asynchronous = true; // Subscribe to the underlying for the underlying position // Set the data normalization mode to raw for strike price comparability _index = AddIndex("SPX").Symbol; // Requesting option data and filter for the hedge candidates var option = AddIndexOption(_index); _optionSymbol = option.Symbol; option.SetFilter(OptionFilter); // Set scheduled event to buy a hedge option contract at market open to eliminate the intra-day movement Schedule.On( DateRules.EveryDay(_index), TimeRules.AfterMarketOpen(_index, 1), BuyHedgeContract ); // Set a scheduled event to sell the hedge contract before market close, since we want to earn from inter-day movement // Leave 2 minutes contingency to fill Schedule.On( DateRules.EveryDay(_index), TimeRules.BeforeMarketClose(_index, 2), SellHedgeContract ); } private OptionFilterUniverse OptionFilter(OptionFilterUniverse universe) { // Select the contracts with delta very close to -1 and high open interest // This can effectively hedge most of the price change of the underlying and ensure the liquidity // Make sure the contract is expiring close for its tradbility return universe .IncludeWeeklys() .PutsOnly() .Expiration(2, 7) .Delta(-1m, -0.95m) .OpenInterest(10, 1000); } private void BuyHedgeContract() { if (CurrentSlice.OptionChains.TryGetValue(_optionSymbol, out var chain)) { // Order the underlying if not hold, the order size should match the option contract // Order only if option chain data ready for hedging if (!Portfolio[_index].Invested) { MarketOrder(_index, Securities[_optionSymbol].SymbolProperties.ContractMultiplier); } // Get the contract with delta closest to -1 (lowest possible delta) var contract = chain.MinBy(x =&GT; x.Greeks.Delta); _hedge = contract.Symbol; // Buy 1 deep ITM put with delta close to -1 to eliminate the intraday movement MarketOrder(_hedge, 1); } } private void SellHedgeContract() { // Check if any hedge contract position, if so, liquidate before market close to expose to underlying overnight movement if (_hedge != null) { Liquidate(_hedge); _hedge = null; } } }
```

Framework Algorithm Example

The following example algorithm demonstrating a Gamma Scalping strategy through framework algorithm. It filters SPX options with expiration between 30 to 90 days and open interest between 50 and 1000, due to liquidity concern and lower Gamma fluctuation. Assuming a significant upward trend is expecting, it orders a long straddle strategy from the strike price with Delta-neutral and the highest Gamma to earn the highest profit from upward underlying movement.

C#

```
public class IndexOptionsUniverseFrameworkAlgorithm : QCAlgorithm
{
    public override void Initialize()
```

```
    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);
        // Asynchronous can use computational resources efficiently
        UniverseSettings.Asynchronous = true;

        // Universe selection that select based on liquidity of the option contracts (by open interest)
        AddUniverseSelection(new IndexOptionsUniverseSelectionModel());
        // Custom alpha model that use Delta and Gamma to signal insights
        AddAlpha(new OptionDeltaGammaAlphaModel());
        // To maintain long-short size equal, use a PCM that order a single contract
        SetPortfolioConstruction(new SingleSharePortfolioConstructionModel());
    }
}

class IndexOptionsUniverseSelectionModel : OptionUniverseSelectionModel
{
    // Daily update with the SelectOptionChainSymbols function
    public IndexOptionsUniverseSelectionModel()
            : base(TimeSpan.FromDays(1), SelectOptionChainSymbols) {}

    private static IEnumerable<Symbol> SelectOptionChainSymbols(DateTime utcTime)
    {
        // Select only SPX options as our focus, which is a highly traded volatile index with great upward momentum
        return new[] {QuantConnect.Symbol.Create("SPX", SecurityType.IndexOption, Market.USA)};
    }

    protected override OptionFilterUniverse Filter(OptionFilterUniverse filter)
    {
        // To ensure the liquidity and tradability, make sure the option is not 0-DTE and have a fair open interest
        // A longer TTM will have lower Gamma thus more stable in the local delta-neutral position
        return filter
            .IncludeWeeklys()
            .Expiration(30, 90)
            .OpenInterest(50, 1000);
    }
}

class OptionDeltaGammaAlphaModel : AlphaModel
{
    // A day count variable to control the alpha model only trade once a day
    private int _day = -1;

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = new List<Insight>();

        if (_day != slice.Time.Day)
        {
            // We open position for each option underlying as separate bet
            foreach (var kvp in slice.OptionChains)
            {
                var chain = kvp.Value;

                // For theta-neutral, select the same expiry for both call and put
                var expiry = chain.Min(x => x.Expiry);
                var contracts = chain.Where(x => x.Expiry == expiry).ToList();

                // Calculate delta and gamma per strike price group for later filtering, ensure both call and put
available for option strategy
                var deltaGammaSymbols = new List<(decimal, decimal, Symbol, Symbol)>();
                var strikes = contracts.Select(x => x.Strike)
                    .Where(x => contracts.Count(y => y.Strike == x) == 2)
                    .Distinct();
                foreach (var strike in strikes)
                {
                    // Get both call and put for their aggregated delta and gamma
                    var call = contracts.Single(x => x.Right == OptionRight.Call && x.Strike == strike);
                    var put = contracts.Single(x => x.Right == OptionRight.Put && x.Strike == strike);
                    deltaGammaSymbols.Add((call.Greeks.Delta + put.Greeks.Delta, call.Greeks.Gamma +
put.Greeks.Gamma, call.Symbol, put.Symbol));
                }

                if (deltaGammaSymbols.Count == 0)
                {
                    continue;
                }

                // We want a delta-neutral position, so it is likely to be ATM (like a long straddle)
                // Less than 0.1 p.d. difference is non-significant, which we can risk for better reward
```

```
                // Less than 2d.p. difference is non-significant, which we can risk for better reward
                // Assuming the market direction is up in most scenario, we try to get the strike providing max
overall Gamma
                // Make sure the aggregated gamma to be positive to bet on large uptrend
                // So it will earn more when the price really go up higher and higher, but locally immune for small
noise
                var filtered = deltaGammaSymbols.Where(item => Math.Round(item.Item1, 2) <= 0.01m && item.Item2 >
0).ToList();
                if (filtered.Count == 0)
                {
                    continue;
                }
                var selected = filtered.OrderByDescending(item => item.Item2).First();

                // Provide trade signal and roll the day count to let the position stay for the whole day
                var selectedCall = selected.Item3;
                var selectedPut = selected.Item4;
                insights.AddRange(new[] {
                    Insight.Price(selectedCall, Expiry.EndOfDay, InsightDirection.Up),
                    Insight.Price(selectedPut, Expiry.EndOfDay, InsightDirection.Up)
                });
                _day = slice.Time.Day;
            }
        }

        return insights;
    }
}

class SingleSharePortfolioConstructionModel : PortfolioConstructionModel
{
    public override IEnumerable<PortfolioTarget> CreateTargets(QCAlgorithm algorithm, Insight[] insights)
    {
        var targets = new List<PortfolioTarget>();
        foreach (var insight in insights)
        {
            if (algorithm.Securities[insight.Symbol].IsTradable)
            {
                // Use a whole number target to order the exact number of share for size-matching
                targets.Add(new PortfolioTarget(insight.Symbol, (int) insight.Direction));
            }
        }
        return targets;
    }
}
```

Data Point Attributes

The US Index Options Universe dataset provides `OptionFilterUniverse` , `OptionUniverse` , and `OptionChain` objects.

OptionFilterUniverse Attributes

`OptionFilterUniverse` objects have the following attributes:

OptionUniverse Attributes

`OptionUniverse` objects have the following attributes:

OptionChain Attributes

`OptionChain` objects have the following attributes:

Datasets

AlgoSeek

AlgoSeek is a leading historical intraday US market data provider offering the most comprehensive and detailed market data and analytics products in the financial industry covering Equities, Futures, Options, cash FOREX, and Cryptocurrencies. AlgoSeek data is built for quantitative trading and machine learning. For more information about AlgoSeek, visit algoseek.com .

US Equities

US Equity Options

US Future Options

US Futures

US Index Options

AlgoSeek

US Equities

Introduction

The US Equities dataset by AlgoSeek is survivorship bias-free daily coverage of every stock traded in the US Securities Information Processors (SIP) CTA/UTP feed since 1998. The dataset covers approximately 27,500 securities, starts in January 1998, and is delivered in any resolution from tick to daily. The Data is collected from the full SIP feed via our Equinix co-located servers, including all trades and quotes published to every exchange as well as FINRA. Over-the-Counter (OTC) trades are not included.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the US Equities dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

AlgoSeek is a leading historical intraday US market data provider offering the most comprehensive and detailed market data and analytics products in the financial industry covering Equities, Futures, Options, cash FOREX, and Cryptocurrencies. AlgoSeek data is built for quantitative trading and machine learning. For more information about AlgoSeek, visit algoseek.com .

Getting Started

AlgoSeek is the default US Equities dataset on QuantConnect. The following snippet demonstrates how to request data from the US Equities dataset:

```C#
_aapl = AddEquity("AAPL", Resolution.Daily).Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 1998
Asset Coverage	27,500 US Equities
Data Density	Dense
Resolution	Tick, Second, Minute, Hourly, & Daily
Timezone	New York
Market Hours	Regular and Extended

Requesting Data

To add US Equities data to your algorithm, call the `AddEquity` method. Save a reference to the Equity `Symbol` so you can access the data later in your algorithm.

```
C#
public class USEquityDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2018, 1, 1);
        SetEndDate(2021, 6, 1);
        SetCash(100000);
        // Subscribe to AAPL data
        _symbol = AddEquity("AAPL", Resolution.Minute).Symbol;
    }
}
```

For more information about creating US Equity subscriptions, see Requesting Data or US Equity Universes .

Accessing Data

To get the current US Equities data, index the `Bars` , `QuoteBars` , or `Ticks` properties of the current `Slice` with the Equity `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    // Access data: TradeBar data
    if (slice.Bars.ContainsKey(_symbol))
    {
        var tradeBar = slice.Bars[_symbol];
        Log($"{_symbol} price at {slice.Time}: {tradeBar.Close}");
    }

    // Access data: QuoteBar data
    if (slice.QuoteBars.ContainsKey(_symbol))
    {
        var quoteBar = slice.QuoteBars[_symbol];
        Log($"{_symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    // Access data: Ticks data
    if (slice.Ticks.ContainsKey(_symbol))
    {
        var ticks = slice.Ticks[_symbol];
        foreach (var tick in ticks)
        {
            Log($"{_symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

You can also iterate through all of the data objects in the current `Slice` .

```C#
public override void OnData(Slice slice)
{
    // Iterate all TradeBar received
    foreach (var kvp in slice.Bars)
    {
        var symbol = kvp.Key;
        var tradeBar = kvp.Value;
        Log($"{symbol} price at {slice.Time}: {tradeBar.Close}");
    }

    // Iterate all QuoteBar received
    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        Log($"{symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    // Iterate all Ticks received
    foreach (var kvp in slice.Ticks)
    {
        var symbol = kvp.Key;
        var ticks = kvp.Value;
        foreach (var tick in ticks)
        {
            Log($"{symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

For more information about accessing US Equities data, see Handling Data .

Historical Data

To get historical US Equity data, call the `History` method with the Equity `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
// TradeBar objects
var historyTradeBars = History(_symbol, 100, Resolution.Daily);

// QuoteBar objects
var historyQuoteBars = History<QuoteBar>(_symbol, 100, Resolution.Minute);

// Tick objects
var historyTicks = History<Tick>(_symbol, TimeSpan.FromSeconds(10), Resolution.Tick);
```

For more information about historical data, see History Requests .

Universe Selection

To select a universe of US Equities, see Equity Universes .

Remove Subscriptions

To unsubscribe from a US Equity that you added with the `AddEquity` method, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_symbol);
```

The `RemoveSecurity` method cancels your open orders for the security and liquidates your holdings.

Supported Assets

To view the supported assets in the US Equities dataset, see the Data Explorer . This dataset doesn't include Over-the-Counter

(OTC) stocks.

Example Applications

The US Equities dataset enables you to accurately design Equity trading strategies. Examples include the following strategies:

- Momentum strategies using historical returns on the premise that the momentum will continue
- Value strategies using fundamental factors on the premise that the price of undervalued securities will rise
- Factor investing with periodic rebalancing

Classic Algorithm Example

The following example algorithm buys and holds Apple stock:

```C#
public class USEquityDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Requesting single equity data, since we only trade AAPL
        _symbol = AddEquity("AAPL", Resolution.Daily).Symbol;

        // Historical data
        var history = History(_symbol, 60, Resolution.Daily);
        Debug($"We got {history.Count()} items from our history request");
    }

    public override void OnData(Slice slice)
    {
        // Check if the current slice containing AAPL and if we hold any position
        // As we make use of the most updated price data to decide the order size
        if (slice.ContainsKey(_symbol) && slice[_symbol] != null && !Portfolio.Invested)
        {
            SetHoldings(_symbol, 1);
        }
    }
}
```

Framework Algorithm Example

The following example algorithm buys and holds Apple stock:

```C#
public class USEquityDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        UniverseSettings.Resolution = Resolution.Daily;
        // To select only AAPL, use a manual selection universe
        var symbols = new[] {QuantConnect.Symbol.Create("AAPL", SecurityType.Equity, Market.USA)};
        AddUniverseSelection(new ManualUniverseSelectionModel(symbols));

        // Constant investment signal
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(7), 0.025, null));

        // Invest in all members equally
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }
}
```

Data Point Attributes

The US Equities dataset provides `TradeBar` , `QuoteBar` , and `Tick` objects.

TradeBar Attributes

`TradeBar` objects have the following attributes:

QuoteBar Attributes

`QuoteBar` objects have the following attributes:

Tick Attributes

`Tick` objects have the following attributes:

AlgoSeek

US Equity Options

Introduction

The US Equity Options data by AlgoSeek provides Option data, including prices, strikes, expires, and open interest. The data covers 4,000 Symbols, starts in January 2012, and is delivered on a minute frequency. This dataset is created by monitoring Options Price Reporting Authority (OPRA) data feed, which consolidates last sale and quotation information originating from the national securities exchanges that have been approved by the Securities and Exchange Commission.

This dataset depends on the following datasets:

- US Equity Security Master - data on splits, dividends, and symbol changes for the underlying Equities.
- US Equity Option Universe - data on the available contracts and their daily Greeks and implied volatility values.
- US Equities - trade and quote data on the underlying Equities.

For more information about the US Equity Options dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

AlgoSeek was in 2014 with the goal of providing the highest quality, most accurate, ready-to-use data in the financial data industry. AlgoSeek provides access to Equities, ETFs, ETNs, Equity Indices, Equity Options, Futures, and Future Options for quantitative firms and traders.

Getting Started

The following snippet demonstrates how to request data from the US Equity Options dataset:

```C#
var option = AddOption("GOOG");
_optionSymbol = option.Symbol;
option.SetFilter(-2, +2, 0, 180);
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2012*
Asset Coverage	4,000 Symbols
Data Density	Dense
Resolution	Minute, Hourly, & Daily
Timezone	New York
Market Hours	Regular Only

* Some data is available before this date. In 2012, AlgoSeek started to fetch data from 48 OPRA channels instead of 24, increasing the quality of the data.

Requesting Data

To add US Equity Options data to your algorithm, call the `AddOption` method. Save a reference to the Equity Option `Symbol` so you can access the data later in your algorithm.

```
C#
public class USEquityOptionsDataAlgorithm : QCAlgorithm
{
    private Symbol _optionSymbol;

    public override void Initialize()
    {
        SetStartDate(2020, 6, 1);
        SetEndDate(2021, 6, 1);
        SetCash(100000);
        UniverseSettings.Asynchronous = true;
        // Request GOOG option data
        var option = AddOption("GOOG");
        _optionSymbol = option.Symbol;
        // Set our strike/expiry filter for this option chain
        option.SetFilter(-2, +2, 0, 180);
    }
}
```

The Equity resolution must be less than or equal to the Equity Option resolution. For example, if you set the Equity resolution to minute, then you must set the Equity Option resolution to minute, hour, or daily.

For more information about creating US Equity Option subscriptions, see Requesting Data or Equity Options Universes .

Accessing Data

To get the current US Equity Options data, index the `OptionChains` property of the current `Slice` with the canonical Equity Option `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your Index Option at every time step. To avoid issues, call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    // Get the wanted option chain with the canonical symbol
    if (slice.OptionChains.TryGetValue(_optionSymbol, out var chain))
    {
        // Iterate the option contracts in chain
        foreach (var contract in chain)
        {
            Log($"{contract.Symbol} price at {slice.Time}: {contract.LastPrice}");
        }
    }
}
```

You can also iterate through all of the `OptionChain` objects in the current `Slice` .

```C#
public override void OnData(Slice slice)
{
    // Iterate all option chains of all symbols
    foreach (var kvp in slice.OptionChains)
    {
        var canonicalSymbol = kvp.Key;
        var chain = kvp.Value;
        // Iterate the option contracts in chain
        foreach (var contract in chain)
        {
            Log($"{contract.Symbol} price at {slice.Time}: {contract.LastPrice}");
        }
    }
}
```

For more information about accessing US Equity Options data, see Handling Data .

Historical Data

You can get historical US Equity Options data in an algorithm and the Research Environment.

Historical Data In Algorithms

To get historical US Equity Options data in an algorithm, call the `History` method with the Equity Option contract `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
// TradeBar objects
var historyTradeBars = History(contract.Symbol, 100, Resolution.Minute);

// QuoteBar objects
var historyQuoteBars = History<QuoteBar>(contract.Symbol, 100, Resolution.Minute);

// OpenInterest objects
var historyOpenInterest = History<OpenInterest>(contract.Symbol, 100, Resolution.Minute);
```

For more information about historical data in algorithms, see History Requests .

Historical Data In Research

To get historical US Equity Options data in the Research Environment, call the `History` or `OptionHistory` method. The `History` method returns the price, volume, and open interest history for some given Option contract(s). The `OptionHistory` method returns the price and volume history for the contracts that pass your daily universe filter.

```C#
var qb = new QuantBook();
var option = qb.AddOption("GOOG");
option.SetFilter(-2, 2, 0, 90);
var history = qb.OptionHistory(option.Symbol, new DateTime(2020, 6, 1), new DateTime(2020, 6, 5));

var contracts = history
    .SelectMany(x => x.OptionChains.SelectMany(y => y.Value.Contracts.Keys))
    .Distinct().ToList();
var expiries = contracts.Select(x => x.ID.Date).Distinct().ToList();
var strikes = contracts.Select(x => x.ID.StrikePrice).Distinct().ToList();
```

To get historical data for arbitrary US Equity Option contracts instead of just the that pass your universe filter, call the `History` method like you would in an algorithm, but on the `QuantBook` object. For more information about historical data in the Research Environment, see Key Concepts .

Historical Greeks and IV Data

To get historical data for the Greeks and implied volatility of Equity Options, see the US Equity Option Universe dataset.

Supported Assets

To view the supported assets in the US Equity Options dataset, see the Data Explorer .

Example Applications

The US Equity Options dataset enables you to accurately design Option strategies. Examples include the following strategies:

- Buying put Options to hedge against downward price movement in positive Equity positions
- Exploiting arbitrage opportunities that arise when the price of Option contracts deviates from their theoretical value

Classic Algorithm Example

The following example algorithm subscribes to Google Options that fall within two strikes of the underlying stock price and expire within seven days. Within this Option chain, the algorithm buys the call Option contract that has the furthest expiry and has its strike price closest to the underlying stock price. When the contract expires, the algorithm rolls over to the next contract that meets this criteria.

```C#
public class USEquityOptionsDataAlgorithm : QCAlgorithm
{
    private Symbol _underlying, _optionSymbol;
    private OptionContract? _contract = null;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        UniverseSettings.Asynchronous = true;
        // Requesting data
        _underlying = AddEquity("GOOG").Symbol;
        var option = AddOption("GOOG");
        _optionSymbol = option.Symbol;
        // To speculate trade the underlying with a low cost, filter for the ATM calls that expiring in the current
week
        // -2/+2 strike buffer is given for small price change
        option.SetFilter((u) => u.IncludeWeeklys().CallsOnly().Strikes(-2, +2).Expiration(0, 7));
    }

    public override void OnData(Slice slice)
    {
        // If the Option contract is exercised, close the underlying position.
        if (Portfolio[_underlying].Invested)
        {
            Liquidate(_underlying);
        }
        // If the Option contract expires, rollover to the next contract.
        if (_contract != null && !Portfolio[_contract.Symbol].Invested)
        {
            _contract = null;
        }
        // Select with the lastest option chain data only
        if (_contract == null && slice.OptionChains.TryGetValue(_optionSymbol, out var chain))
        {
            // Select the call contracts with the furthest expiration (week end)
            var furthestExpiry = chain.Max(c => c.Expiry);
            var furthestExpiryCalls = chain.Where(c => c.Expiry == furthestExpiry);
            // Get the ATM call for speculate trade with low cost and limited loss
            _contract = furthestExpiryCalls.OrderByDescending(x => Math.Abs(chain.Underlying.Price -
x.Strike)).Last();
            MarketOrder(_contract.Symbol, 1);
        }
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            // Historical data
            var history = History(security.Symbol, 100, Resolution.Minute);
            Debug($"We got {history.Count()} from our history request for {security.Symbol}");
        }
    }
}
```

Framework Algorithm Example

The following example algorithm buys a call Option contract for Google that falls within one strike of the underlying stock price and expires within seven days. When the contract expires, the algorithm rolls over to the next contract that meets this criteria.

```C#
public class USEquityOptionsDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        UniverseSettings.Asynchronous = true;
        // Requesting data
        SetUniverseSelection(new EarliestExpiringWeeklyAtTheMoneyCallOptionUniverseSelectionModel());

        SetAlpha(new ConstantOptionsAlphaModel());
```

```
        SetAlpha(new ConstantOptionsAlphaModel());

        SetPortfolioConstruction(new SingleSharePortfolioConstructionModel());
    }
}

class EarliestExpiringWeeklyAtTheMoneyCallOptionUniverseSelectionModel : OptionUniverseSelectionModel
{
    // Daily update with the SelectOptionChainSymbols function
    public EarliestExpiringWeeklyAtTheMoneyCallOptionUniverseSelectionModel()
            : base(TimeSpan.FromDays(1), SelectOptionChainSymbols) {}

    private static IEnumerable<Symbol> SelectOptionChainSymbols(DateTime utcTime)
    {
        // Select only GOOG options as our focus
        return new[] {QuantConnect.Symbol.Create("GOOG", SecurityType.Option, Market.USA)};
    }

    protected override OptionFilterUniverse Filter(OptionFilterUniverse filter)
    {
        // To speculate trade the underlying with a low cost, filter for the ATM calls that expiring in the current
week
        // -2/+2 strike buffer is given for small price change
        return filter
            .Strikes(-1, -1)
            .Expiration(0, 7)
            .WeeklysOnly()
            .CallsOnly();
    }
}

class ConstantOptionsAlphaModel : AlphaModel
{
    private Symbol? _underlying = null;
    private OptionContract? _contract = null;

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = new List<Insight>();

        // Liquidate the underlying if the option is being exercised/assigned
        if (algorithm.Portfolio[_underlying].Invested)
        {
            insights.Add(Insight.Price(_underlying, TimeSpan.FromDays(7), InsightDirection.Flat));
        }

        if (_contract != null && algorithm.Portfolio[_contract.Symbol].Invested)
        {
            return insights;
        }

        // Get the ATM call for speculate trade with low cost and limited loss that expires at week end
        foreach (var kvp in slice.OptionChains)
        {
            var chain = kvp.Value;
            var expiry = chain.Max(x => x.Expiry);
            _contract = chain.Where(x => x.Expiry == expiry)
                .OrderBy(x => Math.Abs(x.Strike - x.UnderlyingLastPrice))
                .First();
            insights.Add(Insight.Price(_contract.Symbol, _contract.Expiry + TimeSpan.FromDays(1),
InsightDirection.Up));
        }
        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            if (security.Type == SecurityType.Equity)
            {
                _underlying = security.Symbol;
            }
            else {
                // Historical data
                var history = algorithm.History(security.Symbol, 100, Resolution.Minute);
                algorithm.Debug($"We got {history.Count()} from our history request for {security.Symbol}");
            }
        }
    }
}
```

```
class SinglePortfolioConstructionModel : PortfolioConstructionModel
{
    public override IEnumerable<PortfolioTarget> CreateTargets(QCAlgorithm algorithm, Insight[] insights)
    {
        var targets = new List<PortfolioTarget>();
        foreach (var insight in insights)
        {
            if (algorithm.Securities[insight.Symbol].IsTradable)
            {
                targets.Add(new PortfolioTarget(insight.Symbol, (int) insight.Direction));
            }
        }
        return targets;
    }
}
```

Data Point Attributes

The US Equity Options dataset provides `TradeBar` , `QuoteBar` , and `OpenInterest` objects.

TradeBar Attributes

`TradeBar` objects have the following attributes:

QuoteBar Attributes

`QuoteBar` objects have the following attributes:

OpenInterest Attributes

`OpenInterest` objects have the following attributes:

AlgoSeek

US Future Options

Introduction

The US Future Options dataset by AlgoSeek provides Option data on US Future contracts, including prices, strikes, and expires. The data covers 15 Monthly Future contracts, starts in January 2012, and is delivered on a minute frequency. This dataset is created by monitoring the trading activity on the CME, CBOT, NYMEX, and COMEX markets.

This dataset depends on the following datasets:

- US Future Option Universe - data on the available US Future Options contracts and their current Open Interest.
- US Futures Security Master - data to construct continuous Futures.
- US Future Universe - data on the available US Future contracts, their daily trading volume, and Open Interest.
- US Futures - data on the underlying Futures contracts, including price, volume, Open Interest, and expiry.

For more information about the US Future Options dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

AlgoSeek is a leading historical intraday US market data provider offering the most comprehensive and detailed market data and analytics products in the financial industry covering equities, futures, options, cash forex, and cryptocurrencies. AlgoSeek data is built for quantitative trading and machine learning. For more information about AlgoSeek, visit algoseek.com .

Getting Started

The following snippet demonstrates how to request data from the US Future Options dataset:

```C#
var future = AddFuture(Futures.Metals.Gold, Resolution.Minute);
future.SetFilter(0, 90);
AddFutureOption(future.Symbol, universe => universe.Strikes(-5, +5));
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2012
Asset Coverage	15 Monthly Future Contracts. Standard expires only*.
Data Density	Dense
Resolution	Minute, Hourly, & Daily
Timezone	New York
Market Hours	Regular and Extended

* No weeklies or 0DTE contracts.

Requesting Data

To add US Future Options data to your algorithm, call the `AddFutureOption` method.

```
C#
public class FutureOptionDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2020, 1, 28);
        SetEndDate(2020, 6, 1);
        SetCash(100000);
        UniverseSettings.Asynchronous = true;
        var future = AddFuture(Futures.Metals.Gold, Resolution.Minute);
        future.SetFilter(0, 90);
        AddFutureOption(future.Symbol, universe => universe.Strikes(-5, +5));
    }
}
```

The Future resolution must be less than or equal to the Future Option resolution. For example, if you set the Future resolution to minute, then the Future Option resolution must be minute, hour, or daily.

For more information about creating Future Options subscriptions, see Requesting Data or Future Options Universes .

Accessing Data

To get the current Future Options data, iterate through the `OptionChains` property of the current `Slice` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your Future Options at every time step.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.OptionChains)
    {
        var canonicalFOPSymbol = kvp.Key;
        var chain = kvp.Value;
        foreach (var contract in chain)
        {
            Log($"{contract.Symbol} price at {slice.Time}: {contract.LastPrice}");
        }
    }
}
```

You can also iterate through the `FuturesChains` in the current `Slice` first.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.FuturesChains)
    {
        var continuousContractSymbol = kvp.Key;
        var futuresChain = kvp.Value;

        // Select a Future Contract and create its canonical FOP Symbol
        var futuresContract = futuresChain.First();
        var canonicalFOPSymbol = QuantConnect.Symbol.CreateCanonicalOption(futuresContract.Symbol);
        if (slice.OptionChains.TryGetValue(canonicalFOPSymbol, out var optionChain))
        {
            foreach (var fopContract in optionChain)
            {
                Log($"{fopContract.Symbol} price at {slice.Time}: {fopContract.LastPrice}");
            }
        }
    }
}
```

For more information about accessing Future Options data, see Handling Data .

Historical Data

You can get historical US Future Options data in an algorithm and the Research Environment.

Historical Data In Algorithms

To get historical US Future Options data in an algorithm, call the `History` method with the Future Option contract `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
// TradeBar objects
var historyTradeBars = History(contract.Symbol, 100, Resolution.Minute);

// QuoteBar objects
var historyQuoteBars = History<QuoteBar>(contract.Symbol, 100, Resolution.Minute);

// OpenInterest objects
var historyOpenInterest = History<OpenInterest >(contract.Symbol, 100, Resolution.Minute);
```

For more information about historical data in algorithms, see History Requests .

Historical Data In Research

To get historical US Future Options data in the Research Environment, call the `History` or `OptionHistory` method. The `History` method returns the price, volume, and open interest history for some given Future Option contract(s). The `OptionHistory` method returns the price and volume history for the contracts that pass your daily universe filter.

```C#
var qb = new QuantBook();
var future = qb.AddFuture(Futures.Indices.SP500EMini);
var startDate = new DateTime(2024, 1, 2);
var futureContractSymbol = qb.FutureChainProvider.GetFutureContractList(future.Symbol, startDate)
    .OrderBy(x => x.ID.Date)
    .First();
var history = qb.OptionHistory(
    futureContractSymbol, startDate, futureContractSymbol.ID.Date, Resolution.Hour
);

var contracts = history
    .SelectMany(x => x.OptionChains.SelectMany(y => y.Value.Contracts.Keys))
    .Distinct().ToList();
var expiries = contracts.Select(x => x.ID.Date).Distinct().ToList();
var strikes = contracts.Select(x => x.ID.StrikePrice).Distinct().ToList();
```

To get historical data for arbitrary US Equity Option contracts instead of just the that pass your universe filter, call the `History` method like you would in an algorithm, but on the `QuantBook` object. For more information about historical data in the Research Environment, see Key Concepts .

Supported Assets

The following list shows the available (15) Futures Options:

- `Futures.Energy.CrudeOilWTI` : Crude Oil WTI Futures (NYMEX: LO | Underlying: CL)
- `Futures.Energy.Gasoline` : Gasoline RBOB Futures (NYMEX: OB | Underlying: RB)
- `Futures.Energy.HeatingOil` : Heating Oil Futures (NYMEX: OH | Underlying: HO)
- `Futures.Energy.NaturalGas` : Natural Gas Futures (NYMEX: ON | Underlying: NG)
- `Futures.Financials.Y10TreasuryNote` : 10Y U.S. Treasury Note Futures (CBOT: OZN | Underlying: ZN)
- `Futures.Financials.Y2TreasuryNote` : 2Y U.S. Treasury Note Futures (CBOT: OZT | Underlying: ZT)

- `Futures.Financials.Y30TreasuryBond` : 30Y U.S. Treasury Bond Futures (CBOT: OZB | Underlying: ZB)
- `Futures.Grains.Corn` : Corn Futures (CBOT: OZC | Underlying: ZC)
- `Futures.Grains.Soybeans` : Soybeans Futures (CBOT: OZS | Underlying: ZS)
- `Futures.Grains.Wheat` : Default wheat contract is SRWWheat (CBOT: OZW | Underlying: ZW)
- `Futures.Indices.NASDAQ100EMini` : E-mini NASDAQ 100 Futures (CME: NQ)
- `Futures.Indices.SP500EMini` : E-mini S&P 500 Futures (CME: ES)
- `Futures.Metals.Copper` : Copper Futures (COMEX: HXE | Underlying: HG)
- `Futures.Metals.Gold` : Gold Futures (COMEX: OG | Underlying: GC)
- `Futures.Metals.Silver` : Silver Futures (COMEX: SO | Underlying: SI)

Example Applications

The US Future Options dataset enables you to accurately design Future Option strategies. Examples include the following strategies:

- Selling out of the money Future Option contracts to collect the premium that the Option buyer pays
- Buying put Options to hedge against downward price movement in Future contracts you bought
- Exploiting arbitrage opportunities that arise when the price of Option contracts deviates from their theoretical value

Classic Algorithm Example

The following example demonstrates a weekly-renewing covered call strategy to collect credit of selling the option. It filters the ATM call contract that expires within the current week at week start using `SetFilter` filtering function.

```C#
public class FutureOptionExampleAlgorithm : QCAlgorithm
{
    private Future _underlying;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(5000000);
        // Subscribe the underlying since the updated price is needed for filtering
        _underlying = AddFuture(Futures.Indices.SP500EMini,
            extendedMarketHours: true,
            dataMappingMode: DataMappingMode.OpenInterest,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            contractDepthOffset: 0);
        // Filter the underlying continuous Futures to narrow the FOP spectrum
        _underlying.SetFilter(0, 182);
        // Filter for the current-week-expiring calls to formulate a covered call that expires at the end of week
        AddFutureOption(_underlying.Symbol, (u) => u.IncludeWeeklys().CallsOnly().Expiration(0, 5));
    }

    public override void OnData(Slice slice)
    {
        // Create canonical symbol for the mapped future contract, since option chains are mapped by canonical symbol
        var symbol = QuantConnect.Symbol.CreateCanonicalOption(_underlying.Mapped);

        // Get option chain data for the mapped future, as both the underlying and FOP have the highest liquidity
among all other contracts
        if (!Portfolio.Invested &&
            slice.OptionChains.TryGetValue(symbol, out var chain))
        {
            // Obtain the ATM call that expires at the end of week, such that both underlying and the FOP expires the
same time
            var expiry = chain.Max(x => x.Expiry);
            var atmCall = chain.Where(x => x.Expiry == expiry)
                .OrderBy(x => Math.Abs(x.Strike - x.UnderlyingLastPrice))
                .First();

            // Use abstraction method to order a covered call to avoid manual error
            var optionStrategy = OptionStrategies.CoveredCall(symbol, atmCall.Strike, expiry);
            Buy(optionStrategy, 1);
        }
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            if (security.Type == SecurityType.FutureOption)
            {
                // Historical data
                var history = History(security.Symbol, 10, Resolution.Minute);
            }
        }
    }
}
```

Framework Algorithm Example

The following example demonstrates a daily renewing speculation trade on upward movement of Gold with FOP using algorithm framework. Using FOP, traders can achieve lower cost to get similar absolute profit, given the FOP is selected to be deep ITM with high Delta value.

```C#
public class FutureOptionDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        // Override an option universe selection model to select the FOPs
        AddUniverseSelection(new GoldFOPUniverseSelectionModel(this));
        // Handle the FOP trading logic in an alpha model
```

```
        SetAlpha(new ConstantFutureOptionsAlphaModel());
        // To order a single contract per insight, use a custom portfolio construction model
        SetPortfolioConstruction(new SingleSharePortfolioConstructionModel());
    }
}

public class GoldFOPUniverseSelectionModel : OptionUniverseSelectionModel
{
    // Daily renewed universe since option contract list is updated on a daily basis
    public GoldFOPUniverseSelectionModel(QCAlgorithm algorithm)
        : base(TimeSpan.FromDays(1), _ => OptionChainSymbolSelector(algorithm, _))
    {
    }

    private static IEnumerable<Symbol> OptionChainSymbolSelector(QCAlgorithm algorithm, DateTime utcTime)
    {
        // Create the underlying symbol to get the FOP contracts later
        var futureSymbol = QuantConnect.Symbol.Create(Futures.Metals.Gold, SecurityType.Future, Market.COMEX);
        // Get all gold FOP contracts of the front month gold contract expiring within 3 months, since their
liquidity is the highest
        return algorithm.FutureChainProvider.GetFutureContractList(futureSymbol, algorithm.Time)
            .Where(futureContractSymbol => futureContractSymbol.ID.Date <= algorithm.Time + TimeSpan.FromDays(90))
            .Select(futureContractSymbol => QuantConnect.Symbol.CreateCanonicalOption(futureContractSymbol));
    }

    protected override OptionFilterUniverse Filter(OptionFilterUniverse optionFilterUniverse)
    {
        // Filter for the ATM calls that expires latest (same expiry as the underlying)
        // +/-5 strike range buffer for price movement
        return optionFilterUniverse.BackMonth().Strikes(-5, +5).CallsOnly();
    }
}


class ConstantFutureOptionsAlphaModel : AlphaModel
{
    // A dictionary to cache the mapped FOP of the underlying Future for filtering wanted option chain data and check
if invested
    private Dictionary<Symbol, OptionContract> optionContractByUnderlyingFutureContract = new Dictionary<Symbol,
OptionContract>();

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = new List<Insight>();

        foreach (var kvp in slice.OptionChains)
        {
            // Liquidate underlying Future contract after Option assignment
            var underlyingFutureContract = kvp.Key.Underlying;
            if (algorithm.Portfolio[underlyingFutureContract].Invested)
            {
                algorithm.Insights.Cancel(new[] { underlyingFutureContract });
                optionContractByUnderlyingFutureContract.Remove(underlyingFutureContract);
            }

            // Do not repeatly invest in the same underlying's FOP
            var chain = kvp.Value.Where(contract => algorithm.Securities[contract.Symbol].IsTradable);
            if (chain.Count() == 0 || optionContractByUnderlyingFutureContract.ContainsKey(underlyingFutureContract))
            {
                continue;
            }

            // Select the Option contract with the lowest strike price to speculate trade the underlying with lowest
cost and highest delta
            var contract = chain.MinBy(contract => contract.Strike);
            insights.Add(Insight.Price(contract.Symbol, contract.Expiry.AddDays(1), InsightDirection.Up));
            optionContractByUnderlyingFutureContract.Add(kvp.Key.Underlying, contract);
        }
        return insights;
    }
}


class SingleSharePortfolioConstructionModel : PortfolioConstructionModel
{
    public override IEnumerable<PortfolioTarget> CreateTargets(QCAlgorithm algorithm, Insight[] insights)
    {
        var targets = new List<PortfolioTarget>();
        foreach (var insight in insights)
        {
            if (algorithm.Securities[insight.Symbol].IsTradable)
```

```
                {
                    // Use integer target to create a portfolio target to trade a single contract
                    targets.Add(new PortfolioTarget(insight.Symbol, (int) insight.Direction));
                }
            }
            return targets;
        }
    }
```

Data Point Attributes

The US Future Options dataset provides `TradeBar` , `QuoteBar` , and `OpenInterest` objects.

TradeBar Attributes

`TradeBar` objects have the following attributes:

QuoteBar Attributes

`QuoteBar` objects have the following attributes:

OpenInterest Attributes

`OpenInterest` objects have the following attributes:

AlgoSeek

US Futures

Introduction

The US Futures dataset by AlgoSeek provides Futures data, including price, volume, open interest, and expiry. The data covers the 157 most liquid contracts, starts in May 2009, and is delivered on any frequency from tick to daily. This dataset is created by monitoring the trading activity on the CFE *, CBOT, CME, COMEX, NYMEX, and ICE *.

This dataset also depends on the US Futures Security Master because the US Futures Security Master dataset contains information to construct continuous Futures.

For more information about the US Futures dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

AlgoSeek is a leading historical intraday US market data provider offering the most comprehensive and detailed market data and analytics products in the financial industry covering equities, futures, options, cash forex, and cryptocurrencies. AlgoSeek data is built for quantitative trading and machine learning. For more information about AlgoSeek, visit algoseek.com .

Getting Started

The following snippet demonstrates how to request data from the US Futures dataset:

```C#
var future = AddFuture(Futures.Metals.Gold);
future.SetFilter(0, 90);
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	May 2009
Asset Coverage	157 Futures
Data Density	Dense
Resolution	Tick, Second, Minute, Hour, & Daily
Timezone	• Chicago (CFE, CME or CBOT) • New York (ICE, NYMEX or COMEX)
Market Hours	Regular and Extended

This dataset only includes Sugar (SB) from the ICE exchange.

This data provider doesn't include a live data feed for the CFE market (VIX).

LBS (Random Length Lumber) was deprecated for LBR (Lumber).

Volume Discrepancies

The volume of the daily trade bars from this dataset is different from the daily volume that CME and other platforms like Yahoo Finance report because the CME includes pit trades in their daily volume calculation. In contrast, the volume of the daily trade bars in this dataset (from AlgoSeek) doesn't include pit trades because pit trades are over-the-counter, which algorithms can't trade.

Another discrepancy occurs from the start and end times of the daily bars. Daily bars from the CME and Yahoo Finance span from 5 PM Central Time (CT) to 4 PM CT on the following day. In contrast, QuantConnect consolidates daily bars from from 12 AM Eastern Time (ET) to 12 AM ET the following day. Therefore, to calculate the daily volume in the same way that CME does (excluding the pit trades), sum the volume of intraday bars that span 5 PM CT to 4 PM CT on the following day. For an example implementation, see Examples .

Requesting Data

To add US Futures data to your algorithm, call the `AddFuture` method. Save a reference to the Future so you can access the data later in your algorithm.

```
C#
public class USFuturesDataAlgorithm : QCAlgorithm
{
    private Symbol _futureSymbol;

    public override void Initialize()
    {
        SetStartDate(2013, 12, 20);
        SetEndDate(2014, 2, 20);
        SetCash(1000000);
        UniverseSettings.Asynchronous = true;
        // Request Gold Future data
        var future = AddFuture(Futures.Metals.Gold);
        // Set filter to obtain the contracts expire within 90 days
        future.SetFilter(0, 90);
        _futureSymbol = future.Symbol;
    }
}
```

For more information about creating Future subscriptions, see Requesting Data or Futures Universes .

Accessing Data

To get the current US Futures data, index the `Bars` , `QuoteBars` , or `Ticks` properties of the current `Slice` with the Future `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```C#
public override void OnData(Slice slice)
{
    // Access data: TradeBar data
    if (slice.Bars.TryGetValue(_futureSymbol, out var tradeBar))
    {
        Log($"{_futureSymbol} price at {slice.Time}: {tradeBar.Close}");
    }

    // Access data: QuoteBar data
    if (slice.QuoteBars.TryGetValue(_futureSymbol, out var quoteBar))
    {
        Log($"{_futureSymbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    // Access data: Ticks data
    if (slice.Ticks.TryGetValue(_futureSymbol, out var ticks))
    {
        foreach (var tick in ticks)
        {
            Log($"{_futureSymbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

You can also iterate through all of the data objects in the current `Slice` .

```C#
public override void OnData(Slice slice)
{
    // Iterate all TradeBar received
    foreach (var kvp in slice.Bars)
    {
        var symbol = kvp.Key;
        var tradeBar = kvp.Value;
        Log($"{symbol} price at {slice.Time}: {tradeBar.Close}");
    }

    // Iterate all QuoteBar received
    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        Log($"{symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    // Iterate all Ticks received
    foreach (var kvp in slice.Ticks)
    {
        var symbol = kvp.Key;
        var ticks = kvp.Value;
        foreach (var tick in ticks)
        {
            Log($"{symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

For more information about accessing US Futures data, see Handling Data .

Historical Data

You can get historical US Futures data in an algorithm and the Research Environment.

Historical Data In Algorithms

To get historical US Futures data in an algorithm, call the `History` method with the canonical Futures `Symbol` or a Futures contract `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
// TradeBar objects
var contractHistoryTradeBars = History(contract.Symbol, 100, Resolution.Minute);
var continuousHistoryTradeBars = History(
    symbols: new[] {_futureSymbol},
    start: Time - TimeSpan.FromDays(15),
    end: Time,
    resolution: Resolution.Minute,
    fillForward: false,
    extendedMarketHours: false,
    dataMappingMode: DataMappingMode.OpenInterest,
    dataNormalizationMode: DataNormalizationMode.Raw,
    contractDepthOffset: 0);

// QuoteBar objects
var contractHistoryQuoteBars = History<QuoteBar>(contract.Symbol, 100, Resolution.Minute);
var continuousHistoryQuoteBars = History<QuoteBar>(_futureSymbol, 100, Resolution.Minute);

// Tick objects
var contractHistoryTicks = History<Tick>(contract.Symbol, TimeSpan.FromSeconds(10), Resolution.Tick);
var continuousHistoryTicks = History<Tick>(_futureSymbol, TimeSpan.FromSeconds(10), Resolution.Tick);
```

For more information about historical data in algorithms, see History Requests . For more information about the price adjustments for continuous contracts, see Continous Contracts .

Historical Data In Research

To get historical US Futures data in the Research Environment for an entire Futures chain, call the `FutureHistory` method with the canonical Future `Symbol` .

```C#
var qb = new QuantBook();
var future = qb.AddFuture(Futures.Metals.Gold);
future.SetFilter(0, 90);
var history = qb.FutureHistory(future.Symbol, new DateTime(2020, 6, 1), new DateTime(2020, 6, 5));

var contracts = history.SelectMany(x => x.OptionChains.SelectMany(y => y.Value.Contracts.Keys)).Distinct().ToList();
var expiries = contracts.Select(x => x.ID.Date).Distinct().ToList();
```

To get historical data for a single US Futures contract or the continuous Futures contract, call the `History` method like you would in an algorithm but on the `QuantBook` object. For more information about historical data in the Research Environment, see Futures .

Supported Assets

The following list shows the available (157) Futures:

- `Futures.Currencies.AUD` : Australian Dollar Futures (CME: 6A)
- `Futures.Currencies.AUDCAD` : Australian Dollar/Canadian Dollar Futures (CME: ACD)
- `Futures.Currencies.AUDJPY` : Australian Dollar/Japanese Yen Futures (CME: AJY)
- `Futures.Currencies.AUDNZD` : Australian Dollar/New Zealand Dollar Futures (CME: ANE)
- `Futures.Currencies.BRL` : Brazillian Real Futures (CME: 6L)
- `Futures.Currencies.BTC` : Bitcoin Futures (CME: BTC)
- `Futures.Currencies.BTICMicroBTC` : BTIC on Micro Bitcoin Futures (CME: MIB)
- `Futures.Currencies.BTICMicroEther` : BTIC on Micro Ether Futures (CME: MRB)
- `Futures.Currencies.CAD` : Canadian Dollar Futures (CME: 6C)
- `Futures.Currencies.CADJPY` : Canadian Dollar/Japanese Yen Futures (CME: CJY)
- `Futures.Currencies.CHF` : Swiss Franc Futures (CME: 6S)
- `Futures.Currencies.ETH` : Ether Futures (CME: ETH)

- `Futures.Currencies.EUR` : Euro FX Futures (CME: 6E)
- `Futures.Currencies.EURAUD` : Euro/Australian Dollar Futures (CME: EAD)
- `Futures.Currencies.EURCAD` : Euro/Canadian Dollar Futures (CME: ECD)
- `Futures.Currencies.EuroFXEmini` : E-mini Euro FX Futures (CME: E7)
- `Futures.Currencies.EURSEK` : Euro/Swedish Krona Futures (CME: ESK)
- `Futures.Currencies.GBP` : British Pound Futures (CME: 6B)
- `Futures.Currencies.JapaneseYenEmini` : E-mini Japanese Yen Futures (CME: J7)
- `Futures.Currencies.JPY` : Japanese Yen Futures (CME: 6J)
- `Futures.Currencies.MicroAUD` : Micro AUD/USD Futures (CME: M6A)
- `Futures.Currencies.MicroBTC` : Micro Bitcoin Futures (CME: MBT)
- `Futures.Currencies.MicroCAD` : Micro USD/CAD Futures (CME: M6C)
- `Futures.Currencies.MicroCADUSD` : Micro CAD/USD Futures (CME: MCD)
- `Futures.Currencies.MicroCHF` : Micro CHF/USD Futures (CME: MSF)
- `Futures.Currencies.MicroEther` : Micro Ether Futures (CME: MET)
- `Futures.Currencies.MicroEUR` : Micro EUR/USD Futures (CME: M6E)
- `Futures.Currencies.MicroGBP` : Micro GBP/USD Futures (CME: M6B)
- `Futures.Currencies.MicroINRUSD` : Micro INR/USD Futures (CME: MIR)
- `Futures.Currencies.MicroJPY` : Micro JPY/USD Futures (CME: MJY)
- `Futures.Currencies.MicroUSDCHF` : Micro USD/CHF Futures (CME: M6S)
- `Futures.Currencies.MicroUSDCNH` : Micro USD/CNH Futures (CME: MNH)
- `Futures.Currencies.MicroUSDJPY` : Micro USD/JPY Futures (CME: M6J)
- `Futures.Currencies.MXN` : Mexican Peso Futures (CME: 6M)
- `Futures.Currencies.NZD` : New Zealand Dollar Futures (CME: 6N)
- `Futures.Currencies.RUB` : Russian Ruble Futures (CME: 6R)
- `Futures.Currencies.StandardSizeUSDOffshoreRMBCNH` : Standard-Size USD/Offshore RMB (CNH) Futures (CME: CNH)
- `Futures.Currencies.ZAR` : South African Rand Futures (CME: 6Z)
- `Futures.Energy.ArgusLLSvsWTIArgusTradeMonth` : Argus LLS vs. WTI (Argus) Trade Month Futures (NYMEX: AE5)
- `Futures.Energy.ArgusPropaneFarEastIndex` : Argus Propane Far East Index Futures (NYMEX: A7E)
- `Futures.Energy.ArgusPropaneSaudiAramco` : Argus Propane (Saudi Aramco) Futures (NYMEX: A9N)
- `Futures.Energy.BrentCrudeOilVsDubaiCrudeOilPlatts` : Brent Crude Oil vs. Dubai Crude Oil (Platts) Futures (NYMEX: ADB)
- `Futures.Energy.BrentLastDayFinancial` : Brent Last Day Financial Futures (NYMEX: BZ)
- `Futures.Energy.ChicagoEthanolPlatts` : Chicago Ethaanol (Platts) Futures (NYMEX: CU)
- `Futures.Energy.ConwayPropaneOPIS` : Conway Propane (OPIS) Futures (NYMEX: A8K)
- `Futures.Energy.CrudeOilWTI` : Crude Oil WTI Futures (NYMEX: CL)
- `Futures.Energy.DubaiCrudeOilPlattsFinancial` : Dubai Crude Oil (Platts) Financial Futures (NYMEX: DCB)
- `Futures.Energy.EastWestGasolineSpreadPlattsArgus` : East-West Gasoline Spread (Platts-Argus) Futures (NYMEX: EWG)
- `Futures.Energy.EastWestNaphthaJapanCFvsCargoesCIFNWESpreadPlatts` : East-West Naphtha: Japan C&F vs. Cargoes CIF NWE Spread (Platts) Futures (NYMEX: EWN)
- `Futures.Energy.Ethanol` : Ethanol Futures (CBOT: EH)
- `Futures.Energy.EthanolT2FOBRdamIncludingDutyPlatts` : Ethanol T2 FOB Rdam Including Duty (Platts) Futures (NYMEX: AZ1)
- `Futures.Energy.EuropeanNaphthaPlattsCrackSpread` : European Naphtha (Platts) Crack Spread Futures (NYMEX: EN)
- `Futures.Energy.EuropeanPropaneCIFARAArgus` : European Propane CIF ARA (Argus) Futures (NYMEX: APS)

- `Futures.Energy.EuropeanPropaneCIFARAArgusVsNaphthaCargoesCIFNWEPlatts` : European Propane CIF ARA (Argus) vs. Naphtha Cargoes CIF NWE (Platts) Futures (NYMEX: EPN)
- `Futures.Energy.FreightRouteTC14Baltic` : Freight Route TC14 (Baltic) Futures (NYMEX: FRC)
- `Futures.Energy.Gasoline` : Gasoline RBOB Futures (NYMEX: RB)
- `Futures.Energy.GasolineEurobobOxyNWEBargesArgus` : Gasoline Euro-bob Oxy NWE Barges (Argus) Futures (NYMEX: B7H)
- `Futures.Energy.GroupThreeSuboctaneGasolinePlattsVsRBOB` : Group Three Sub-octane Gasoliine (Platts) vs. RBOB Futures (NYMEX: AA8)
- `Futures.Energy.GroupThreeULSDPlattsVsNYHarborULSD` : Group Three ULSD (Platts) vs. NY Harbor ULSD Futures (NYMEX: AA6)
- `Futures.Energy.GulfCoastCBOBGasolineA2PlattsVsRBOBGasoline` : Gulf Coast CBOB Gasoline A2 (Platts) vs. RBOB Gasoline Futures (NYMEX: CRB)
- `Futures.Energy.GulfCoastHSFOPlattsVsEuropeanThreePointFivePercentFuelOilBargesFOBRdamPlatts` : Gulf Coast HSFO (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures (NYMEX: GCU)
- `Futures.Energy.HeatingOil` : Heating Oil Futures (NYMEX: HO)
- `Futures.Energy.LosAngelesCARBOBGasolineOPISvsRBOBGasoline` : Los Angeles CARBOB Gasoline (OPIS) vs. RBOB Gasoline Futures (NYMEX: AJL)
- `Futures.Energy.LosAngelesCARBDieselOPISvsNYHarborULSD` : Los Angeles CARB Diesel (OPIS) vs. NY Harbor ULSD Futures (NYMEX: AKL)
- `Futures.Energy.LosAngelesJetOPISvsNYHarborULSD` : Los Angeles Jet (OPIS) vs. NY Harbor ULSD Futures (NYMEX: AJS)
- `Futures.Energy.MarsArgusVsWTIFinancial` : Mars (Argus) vs. WTI Financial Futures (NYMEX: AYX)
- `Futures.Energy.MarsArgusVsWTITradeMonth` : Mars (Argus) vs. WTI Trade Month Futures (NYMEX: AYV)
- `Futures.Energy.MicroCrudeOilWTI` : Micro WTI Crude Oil Futures (NYMEX: MCL)
- `Futures.Energy.MicroEuropeanFOBRdamMarineFuelZeroPointFivePercentBargesPlatts` : Micro European FOB Rdam Marine Fuel 0.5% Barges (Platts) Futures (NYMEX: R5O)
- `Futures.Energy.MicroEuropeanThreePointFivePercentOilBargesFOBRdamPlatts` : Micro European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures (NYMEX: MEF)
- `Futures.Energy.MicroGasoilZeroPointOnePercentBargesFOBARAPlatts` : Micro Gasoil 0.1% Barges FOB ARA (Platts) Futures (NYMEX: M1B)
- `Futures.Energy.MicroSingaporeFOBMarineFuelZeroPointFivePercetPlatts` : Micro Singapore FOB Marine Fuel 0.5% (Platts) Futures (NYMEX: S5O)
- `Futures.Energy.MicroSingaporeFuelOil380CSTPlatts` : Micro Singapore Fuel Oil 380CST (Platts) Futures (NYMEX: MAF)
- `Futures.Energy.MiniEuropeanThreePointPercentFiveFuelOilBargesPlatts` : Mini European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures (NYMEX: A0D)
- `Futures.Energy.MiniSingaporeFuelOil180CstPlatts` : Mini Singapore Fuel Oil 180 cst (Platts) Futures (NYMEX: A0F)
- `Futures.Energy.MontBelvieuEthaneOPIS` : Mont Belvieu Ethane (OPIS) Futures (NYMEX: AC0)
- `Futures.Energy.MontBelvieuLDHPropaneOPIS` : Mont Belvieu LDH Propane (OPIS) Futures (NYMEX: B0)
- `Futures.Energy.MontBelvieuNaturalGasolineOPIS` : Mont Belvieu Natural Gasoline (OPIS) Futures (NYMEX: A7Q)
- `Futures.Energy.MontBelvieuNormalButaneOPIS` : Mont Belvieu Normal Butane (OPIS) Futures (NYMEX: AD0)
- `Futures.Energy.NaturalGas` : Natural Gas Futures (NYMEX: NG)
- `Futures.Energy.NaturalGasHenryHubLastDayFinancial` : Natural Gas (Henry Hub) Last-day Financial Futures (NYMEX: HH)
- `Futures.Energy.NaturalGasHenryHubPenultimateFinancial` : Natural Gas (Henry Hub) Penultimate Financial Futures (NYMEX: HP)
- `Futures.Energy.OnePercentFuelOilCargoesFOBNWEPlattsVsThreePointFivePercentFuelOilBargesFOBRdamPlatts` : 1%

Fuel Oil Cargoes FOB NWE (Platts) vs. 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures (NYMEX: FSS)

- `Futures.Energy.PremiumUnleadedGasoline10ppmFOBMEDPlatts` : Premium Unleaded Gasoline 10 ppm FOB MED (Platts) Futures (NYMEX: A3G)
- `Futures.Energy.PropaneNonLDHMontBelvieuOPIS` : Propane Non-LDH Mont Belvieu (OPIS) Futures (NYMEX: A1R)
- `Futures.Energy.RBOBGasolineCrackSpread` : RBOB Gasoline Crack Spread Futures (NYMEX: ARE)
- `Futures.Energy.RBOBGasolineVsEurobobOxyNWEBargesArgusThreeHundredFiftyThousandGallons` : RBOB Gasoline vs. Euro-bob Oxy NWE Barges (Argus) (350,000 gallons) Futures (NYMEX: EXR)
- `Futures.Energy.SingaporeFuelOil380cstPlattsVsEuropeanThreePointFivePercentFuelOilBargesFOBRdamPlatts` : Singapore Fuel Oil 380 cst (Platts) vs. European 3.5% Fuel Oil Barges FOB Rdam (Platts) Futures (NYMEX: EVC)
- `Futures.Energy.SingaporeGasoilPlattsVsLowSulphurGasoilFutures` : Singapore Gasoil (Platts) vs. Low Sulphur Gasoil Futures (NYMEX: AGA)
- `Futures.Energy.SingaporeMogas92UnleadedPlattsBrentCrackSpread` : Singapore Mogas 92 Unleaded (Platts) Brent Crack Spread Futures (NYMEX: D1N)
- `Futures.Energy.ThreePointFivePercentFuelOilBargesFOBRdamPlattsCrackSpread` : 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread Futures (NYMEX: FO)
- `Futures.Energy.ThreePointFivePercentFuelOilBargesFOBRdamPlattsCrackSpread1000mt` : 3.5% Fuel Oil Barges FOB Rdam (Platts) Crack Spread (1000mt) Futures (NYMEX: BOO)
- `Futures.Energy.WTIBrentFinancial` : WTI-Brent Financial Futures (NYMEX: BK)
- `Futures.Energy.WTIFinancial` : WTI Financial Futures (NYMEX: CSX)
- `Futures.Energy.WTIHoustonArgusVsWTITradeMonth` : WTI Houston (Argus) vs. WTI Trade Month Futures (NYMEX: HTT)
- `Futures.Energy.WTIHoustonCrudeOil` : WTI Houston Crude Oil Futures (NYMEX: HCL)
- `Futures.Financials.EuroDollar` : EuroDollar Futures (CME: GE)
- `Futures.Financials.FiveYearUSDMACSwap` : 5-Year USD MAC Swap Futures (CBOT: F1U)
- `Futures.Financials.MicroY10TreasuryNote` : Micro 10-Year Yield Futures (CBOT: 10Y)
- `Futures.Financials.MicroY2TreasuryBond` : Micro 2-Year Yield Futures (CBOT: 2YY)
- `Futures.Financials.MicroY30TreasuryBond` : Micro 30-Year Yield Futures (CBOT: 30Y)
- `Futures.Financials.MicroY5TreasuryBond` : Micro 5-Year Yield Futures (CBOT: 5YY)
- `Futures.Financials.UltraTenYearUSTreasuryNote` : Ultra 10-Year U.S. Treasury Note Futures (CBOT: TN)
- `Futures.Financials.UltraUSTreasuryBond` : Ultra U.S. Treasury Bond Futures (CBOT: UB)
- `Futures.Financials.Y10TreasuryNote` : 10Y U.S. Treasury Note Futures (CBOT: ZN)
- `Futures.Financials.Y2TreasuryNote` : 2Y U.S. Treasury Note Futures (CBOT: ZT)
- `Futures.Financials.Y30TreasuryBond` : 30Y U.S. Treasury Bond Futures (CBOT: ZB)
- `Futures.Financials.Y5TreasuryNote` : 5Y U.S. Treasury Note Futures (CBOT: ZF)
- `Futures.Forestry.Lumber` : Lumber Futures (CME: LBR)
- `Futures.Forestry.RandomLengthLumber` : Random Length Lumber Futures (CME: LBS)
- `Futures.Grains.BlackSeaCornFinanciallySettledPlatts` : Black Sea Corn Financially Settled (Platts) Futures (CBOT: BCF)
- `Futures.Grains.BlackSeaWheatFinanciallySettledPlatts` : Black Sea Wheat Financially Settled (Platts) Futures (CBOT: BWF)
- `Futures.Grains.Corn` : Corn Futures (CBOT: ZC)
- `Futures.Grains.HRWWheat` : KC HRW Wheat Futures (CBOT: KE)
- `Futures.Grains.Oats` : Oats Futures (CBOT: ZO)
- `Futures.Grains.Soybeans` : Soybeans Futures (CBOT: ZS)
- `Futures.Grains.SoybeanMeal` : Soybean Meal Futures (CBOT: ZM)
- `Futures.Grains.SoybeanOil` : Soybean Oil Futures (CBOT: ZL)

- `Futures.Grains.Wheat` : Default wheat contract is SRWWheat (CBOT: ZW)
- `Futures.Indices.BloombergCommodityIndex` : Bloomberg Commodity Index Futures (CBOT: AW)
- `Futures.Indices.Dow30EMini` : E-mini Dow Indu 30 Futures (CBOT: YM)
- `Futures.Indices.DowJonesRealEstate` : Dow Jones Real Estate futures on CME (CME: RX)
- `Futures.Indices.FTSEEmergingEmini` : E-mini FTSE Emerging Index Futures (CME: EI)
- `Futures.Indices.MicroDow30EMini` : Micro E-mini Dow Jones Industrial Average Index Futures (CBOT: MYM)
- `Futures.Indices.MicroNASDAQ100EMini` : Micro E-mini Nasdaq-100 Index Futures (CME: MNQ)
- `Futures.Indices.MicroRussell2000EMini` : Micro E-mini Russell 2000 Index Futures (CME: M2K)
- `Futures.Indices.MicroSP500EMini` : Micro E-mini S&P 500 Index Futures (CME: MES)
- `Futures.Indices.NASDAQ100BiotechnologyEMini` : E-mini Nasdaq-100 Biotechnology Index Futures (CME: BIO)
- `Futures.Indices.NASDAQ100EMini` : E-mini NASDAQ 100 Futures (CME: NQ)
- `Futures.Indices.Nikkei225Dollar` : Nikkei-225 Dollar Futures (CME: NKD)
- `Futures.Indices.Nikkei225YenCME` : Nikkei-225 Yen denominated Futures on CME (CME: NIY)
- `Futures.Indices.Russell1000EMini` : E-mini Russell 1000 futures on CME (CME: RS1)
- `Futures.Indices.Russell2000EMini` : E-mini Russell 2000 Futures (CME: RTY)
- `Futures.Indices.SPGSCICommodity` : S&P-GSCI Commodity Index Futures (CME: GD)
- `Futures.Indices.SP400MidCapEmini` : E-mini S&P MidCap 400 Futures (CME: EMD)
- `Futures.Indices.SP500AnnualDividendIndex` : (CME: SDA)
- `Futures.Indices.SP500EMini` : E-mini S&P 500 Futures (CME: ES)
- `Futures.Indices.TOPIXYEN` : YEN Denominated Topix Index Futures on CME (CME: TPY)
- `Futures.Indices.USDDenominatedIbovespa` : USD-Denominated Ibovespa Index Futures (CME: IBV)
- `Futures.Indices.VIX` : CBOE Volatility Index Futures (CFE: VX)
- `Futures.Meats.FeederCattle` : Feeder Cattle Futures (CME: GF)
- `Futures.Meats.LeanHogs` : Lean Hogs Futures (CME: HE)
- `Futures.Meats.LiveCattle` : Live Cattle Futures (CME: LE)
- `Futures.Metals.AluminiumEuropeanPremiumDutyPaidMetalBulletin` : Aluminium European Premium Duty-Paid (Metal Bulletin) Futures (COMEX: EDP)
- `Futures.Metals.AluminumMWUSTransactionPremiumPlatts25MT` : Aluminum MW U.S. Transaction Premium Platts (25MT) Futures (COMEX: AUP)
- `Futures.Metals.Copper` : Copper Futures (COMEX: HG)
- `Futures.Metals.Gold` : Gold Futures (COMEX: GC)
- `Futures.Metals.MicroGold` : Micro Gold Futures (COMEX: MGC)
- `Futures.Metals.MicroGoldTAS` : Micro Gold TAS Futures (COMEX: MGT)
- `Futures.Metals.MicroPalladium` : Micro Palladium Futures (NYMEX: PAM)
- `Futures.Metals.MicroSilver` : Micro Silver Futures (COMEX: SIL)
- `Futures.Metals.Palladium` : Palladium Futures (NYMEX: PA)
- `Futures.Metals.Platinum` : Platinum Futures (NYMEX: PL)
- `Futures.Metals.Silver` : Silver Futures (COMEX: SI)
- `Futures.Metals.USMidwestDomesticHotRolledCoilSteelCRUIndex` : U.S. Midwest Domestic Hot-Rolled Coil Steel (CRU) Index Futures (NYMEX: HRC)
- `Futures.Softs.Sugar11` : Sugar #11 Futures ICE (ICE: SB)
- `Futures.Softs.Sugar11CME` : Sugar #11 Futures CME (NYMEX: YO)

Example Applications

The US Futures dataset enables you to accurately design Futures strategies. Examples include the following strategies:

- Buying the Futures contract with the most open interest to reduce slippage and market impact
- Trading bull calendar spreads to reduce volatility and margin requirements

Classic Algorithm Example

The following example algorithm buys the Mini Gold Futures contract with the greatest open interest and sells the Micro Gold Futures contract with the greatest open interest. When the open interest of a different contract exceeds the open interest of a contract in the portfolio, the algorithm rebalances the portfolio.

```
C#
public class USFuturesDataAlgorithm : QCAlgorithm
{
    private Future _gold, _microGold;
    private decimal _goldMulitplier, _microGoldMulitplier;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);
        // Setting the continuous contract mapping criteria for both Gold and Micro Gold contracts, since we want to
order the highest liquidity contracts
        Settings.SeedInitialPrices = true;
        _gold = AddFuture(Futures.Metals.Gold,
            extendedMarketHours: true,
            dataMappingMode: DataMappingMode.OpenInterest,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            contractDepthOffset: 0);
        _microGold = AddFuture(Futures.Metals.MicroGold,
            extendedMarketHours: true,
            dataMappingMode: DataMappingMode.OpenInterest,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            contractDepthOffset: 0);
        // The contract multiplier is cached in the Security SymbolProperties property from the symbol properties
database
        _goldMulitplier = _gold.SymbolProperties.ContractMultiplier;
        _microGoldMulitplier = _microGold.SymbolProperties.ContractMultiplier;
    }

    public override void OnData(Slice slice)
    {
        // Make sure to calculate the order size by the most updated price data of both contracts
        if (!Portfolio.Invested && slice.Bars.ContainsKey(_gold.Symbol) && slice.Bars.ContainsKey(_microGold.Symbol))
        {
            // Calculate the order size for $500k
            // Get the quotient after dividing the contract multiplier since the order size must be whole number
            var goldQuantity = Math.Floor(500000m / slice.Bars[_gold.Symbol].Close / _goldMulitplier);
            var microGoldQuantity = Math.Floor(500000m / slice.Bars[_microGold.Symbol].Close / _microGoldMulitplier);

            MarketOrder(_gold.Mapped, goldQuantity);
            MarketOrder(_microGold.Mapped, microGoldQuantity);
        }
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            // Historical data
            var history = History(security.Symbol, 100, Resolution.Minute);
            Debug($"We got {history.Count()} from our history request for {security.Symbol}");
        }
    }
}
```

Framework Algorithm Example

The following example algorithm buys the front-month Mini Gold Futures contract and sells the front-month Micro Gold Futures contract. When the front-month contract changes, the algorithm rebalances the portfolio.

```
C#
public class USFuturesDataAlgorithm : QCAlgorithm
```

```
public class GoldMicroFuturesDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);
        UniverseSettings.Asynchronous = true;
        // Set up an universe selection model that selects the front month contract
        SetUniverseSelection(new FrontMonthFutureUniverseSelectionModel());
        SetAlpha(new ConstantFuturesAlphaModel());
        // A portfolio construction model that only order a single share per insight signal
        SetPortfolioConstruction(new SingleSharePortfolioConstructionModel());
    }
}

class FrontMonthFutureUniverseSelectionModel : FutureUniverseSelectionModel
{
    // Daily updating with select_future_chain_symbols function
    public FrontMonthFutureUniverseSelectionModel()
        : base(TimeSpan.FromDays(1), SelectFutureChainSymbols) {}

    private static IEnumerable<Symbol> SelectFutureChainSymbols(DateTime utcTime)
    {
        //Select gold and micro gold contracts for the strategy need
        return new List<Symbol> {
            QuantConnect.Symbol.Create(Futures.Metals.Gold, SecurityType.Future, Market.COMEX),
            QuantConnect.Symbol.Create(Futures.Metals.MicroGold, SecurityType.Future, Market.COMEX)
        };
    }

    protected override FutureFilterUniverse Filter(FutureFilterUniverse filter)
    {
        // Filter only front month contract for liquidity and most informed information
        return filter.FrontMonth().OnlyApplyFilterAtMarketOpen();
    }
}


class ConstantFuturesAlphaModel : AlphaModel
{
    // Long gold and short micro gold in this strategy
    private Symbol
        _longSymbol = QuantConnect.Symbol.Create(Futures.Metals.Gold, SecurityType.Future, Market.COMEX),
        _shortSymbol = QuantConnect.Symbol.Create(Futures.Metals.MicroGold, SecurityType.Future, Market.COMEX);

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = new List<Insight>();

        if (algorithm.Portfolio.Invested)
        {
            return insights;
        }

        // For both gold and micro gold, select the front month contract (only contract) in the chain
        foreach (var kvp in slice.FutureChains)
        {
            var symbol = kvp.Key;
            var chain = kvp.Value;
            var contract = chain.First();

            // Long gold and short micro gold as planned
            if (symbol == _longSymbol)
            {
                insights.Add(Insight.Price(contract.Symbol, contract.Expiry + TimeSpan.FromDays(1),
InsightDirection.Up));
            }
            else if (symbol == _shortSymbol)
            {
                insights.Add(Insight.Price(contract.Symbol, contract.Expiry + TimeSpan.FromDays(1),
InsightDirection.Down));
            }
        }

        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
```

```
            {
                // Historical data
                var history = algorithm.History(security.Symbol, 100, Resolution.Minute);
                algorithm.Debug($"We got {history.Count()} from our history request for {security.Symbol}");
            }
        }
    }


    class SingleSharePortfolioConstructionModel : PortfolioConstructionModel
    {
        public override IEnumerable<PortfolioTarget> CreateTargets(QCAlgorithm algorithm, Insight[] insights)
        {
            var targets = new List<PortfolioTarget>();
            foreach (var insight in insights)
            {
                if (algorithm.Securities[insight.Symbol].IsTradable)
                {
                    // Single share only using integer portfolio target
                    targets.Add(new PortfolioTarget(insight.Symbol, (int) insight.Direction));
                }
            }
            return targets;
        }
    }
```

Data Point Attributes

The US Futures dataset provides `FuturesChain` , `Future` , and `OpenInterest` objects. To configure the continuous Future settings, use the `DataNormalizationMode` and `DataMappingMode` enumerations.

DataNormalizationMode Values

The `DataNormalizationMode` enumeration has the following values:

DataMappingMode Values

The `DataMappingMode` enumeration has the following values:

Future Attributes

`Future` objects have the following attributes:

FuturesChain Attributes

`FuturesChain` objects have the following attributes:

OpenInterest Attributes

`OpenInterest` objects have the following attributes:

AlgoSeek

US Index Options

Introduction

The US Index Options dataset by AlgoSeek provides Option data, including prices, strikes, expires, and open interest. The data covers European Option contracts for 3 US Indices: SPX, VIX, and NDX. It starts from January 2012 and is delivered on minute resolution. This dataset is created by monitoring the Options Price Reporting Authority (OPRA) data feed, which consolidates last sale and quotation information originating from the national securities exchanges that have been approved by the Securities and Exchange Commission.

The US Index Options dataset depends on the US Index Option Universe dataset because the US Index Options Universe dataset contains information on the available contracts, including their daily Greeks and implied volatility values.

For more information about the US Index Options dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

AlgoSeek is a leading historical intraday US market data provider offering the most comprehensive and detailed market data and analytics products in the financial industry covering Equities, Futures, Options, cash FOREX, and Cryptocurrencies. AlgoSeek data is built for quantitative trading and machine learning. For more information about AlgoSeek, visit algoseek.com .

Getting Started

The following snippet demonstrates how to request data from the US Index Options dataset:

```C#
_indexSymbol = AddIndex("VIX").Symbol;
var option = AddIndexOption(_indexSymbol);
option.SetFilter(-2, 2, 0, 90);
_optionSymbol = option.Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2012
Asset Coverage	7 Index Options
Data Density	Regular
Resolution	Minute, Hourly, & Daily
Timezone	New York
Market Hours	Regular Only

Specification Over Time

According to the SPX Options contract specification , some SPX contracts expire every month and SPXW contracts expires every day. Before 2021, you could only trade SPX contracts with the following expiration dates:

- Expires within the next 4 months
- Expires in September within the next 14 months
- Expires in January, March, or June within the next 2 years
- Expires in December within the next 3 years

During this time, SPXW didn't have 0DTE every day.

`Sources` :

- Cboe Options Exchange to List Three Long-Dated SPX Options Expirations, Beginning November 1, 2021

- S&P 500 Weekly Options Now Expire Five Days a Week

Requesting Data

To add US Index Options data to your algorithm, call the `AddIndexOption` method. Save a reference to the Index Option `Symbol` so you can access the data later in your algorithm.

```
C#
public class IndexOptionsDataAlgorithm : QCAlgorithm
{
    private Symbol _indexSymbol, _standardOptionSymbol, _weeklyOptionSymbol;

    public override void Initialize()
    {
        SetStartDate(2021, 1, 1);
        SetEndDate(2021, 6, 1);
        SetCash(100000);
        UniverseSettings.Asynchronous = true;
        _indexSymbol = AddIndex("VIX").Symbol;

        var standardOption = AddIndexOption(_indexSymbol);
        standardOption.SetFilter(-2, 2, 0, 90);
        _standardOptionSymbol = standardOption.Symbol;

        var weeklyOption = AddIndexOption(_indexSymbol, "VIXW");
        weeklyOption.SetFilter(-2, 2, 0, 90);
        _weeklyOptionSymbol = weeklyOption.Symbol;
    }
}
```

The Index resolution must be less than or equal to the Index Option resolution. For example, if you set the Index resolution to minute, then you must set the Index Option resolution to minute, hour, or daily.

For more information about creating US Index Option subscriptions, see Requesting Data or Index Options Universes .

Accessing Data

To get the current US Index Options data, index the `OptionChains` property of the current `Slice` with the canonical Index Option `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your Index Option at every time step.

```
public override void OnData(Slice slice)
{
    if (slice.OptionChains.ContainsKey(_standardOptionSymbol))
    {
        var standardChain = slice.OptionChains[_standardOptionSymbol];
        foreach (var contract in standardChain)
        {
            Log($"{contract.Symbol} price at {slice.Time}: {contract.LastPrice}");
        }
    }

    if (slice.OptionChains.ContainsKey(_weeklyOptionSymbol))
    {
        var weeklyChain = slice.OptionChains[_weeklyOptionSymbol];
        foreach (var contract in weeklyChain)
        {
            Log($"{contract.Symbol} price at {slice.Time}: {contract.LastPrice}");
        }
    }
}
```

C#

You can also iterate through all of the `OptionChain` objects in the current `Slice` .

```
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.OptionChains)
    {
        var canonicalSymbol = kvp.Key;
        var chain = kvp.Value;
        foreach (var contract in chain)
        {
            Log($"{contract.Symbol} price at {slice.Time}: {contract.LastPrice}");
        }
    }
}
```

C#

For more information about accessing US Index Options data, see Handling Data .

Historical Data

You can get historical US Index Options data in an algorithm and the Research Environment.

Historical Data In Algorithms

To get historical US Index Options data in an algorithm, call the `History` method with the Index Option contract `Symbol` . If there is no data in the period you request, the history result is empty.

```
// TradeBar objects
var historyTradeBars = History(contract.Symbol, 100, Resolution.Minute);

// QuoteBar objects
var historyQuoteBars = History<QuoteBar>(contract.Symbol, 100, Resolution.Minute);

// OpenInterest objects
var historyOpenInterest = History<OpenInterest>(contract.Symbol, 100, Resolution.Minute);
```

C#

For more information about historical data in algorithms, see History Requests .

Historical Data In Research

To get historical US Index Options data in the Research Environment, call the `History` or `OptionHistory` method. The `History` method returns the price, volume, and open interest history for some given Option contract(s). The `OptionHistory` method

returns the price and volume history for the contracts that pass your daily universe filter.

```
C#
var qb = new QuantBook();
var indexSymbol = qb.AddIndex("VIX").Symbol;
var option = qb.AddIndexOption(indexSymbol); // or qb.AddIndexOption(indexSymbol, "VIXW");
option.SetFilter(-2, 2, 0, 90);
var history = qb.OptionHistory(option.Symbol, new DateTime(2020, 6, 1), new DateTime(2020, 6, 5));

var contracts = history
    .SelectMany(x => x.OptionChains.SelectMany(y => y.Value.Contracts.Keys))
    .Distinct().ToList();
var expiries = contracts.Select(x => x.ID.Date).Distinct().ToList();
var strikes = contracts.Select(x => x.ID.StrikePrice).Distinct().ToList();
```

To get historical data for arbitrary US Index Option contracts instead of just the that pass your universe filter, call the `History` method like you would in an algorithm, but on the `QuantBook` object. For more information about historical data in the Research Environment, see Key Concepts .

Historical Greeks and IV Data

To get historical data for the Greeks and implied volatility of Index Options, see the US Index Option Universe dataset.

Supported Assets

The following table shows the available Index Options:

Underlying Index	Underlying Ticker	Target Ticker	Standard Contracts	Weekly Contracts	Tradable on Expiry Day
NASDAQ-100	NDX		✓		
NASDAQ-100	NDX	NDXP	✓		✓
NASDAQ-100	NDX	NQX	✓	✓	✓
Russell 2000	RUT		✓		
Russell 2000	RUT	RUTW		✓	✓
S&P500	SPX		✓		
S&P500	SPX	SPXW		✓	✓
S&P500	VIX		✓		
S&P500	VIX	VIXW		✓	

For more information about each underlying Index, see Supported Indices .

Example Applications

The US Index Options dataset enables you to accurately design strategies for Index Options. Examples include the following strategies:

- Buying VIX call Options to hedge against upcoming volatility
- Buying VIX put Options to capture the natural downward price movement in the VIX index
- Buying SPX put Options to protect against downward price movement in the S&P 500

Classic Algorithm Example

The following example algorithm buys the VIX Index Option Bull Call Spread with the furthest expiration:

C#

```
public class IndexOptionsDataAlgorithm : QCAlgorithm
{
    private Symbol _optionSymbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(200000);
        // Asynchronous can use computational resources efficiently
        UniverseSettings.Asynchronous = true;
        // Filter to get ATM calls expiring in 180 days to form the Bull Call Spread
        var option = AddIndexOption("SPX");
        option.SetFilter((u) => u.CallsOnly().Strikes(-2, +2).Expiration(0, 180));
        _optionSymbol = option.Symbol;
    }

    public override void OnData(Slice slice)
    {
        if (!Portfolio.Invested && IsMarketOpen(_optionSymbol))
        {
            // Make sure getting the updated VIX option chain
            if (slice.OptionChains.TryGetValue(_optionSymbol, out var chain))
            {
                var expiry = chain.Max(x => x.Expiry);
                var callContracts = chain
                    .Where(x => x.Expiry == expiry)
                    .OrderBy(x => x.Strike)
                    .ToList();

                // Need 2 contracts to form a call spread
                if (callContracts.Count < 2)
                {
                    return;
                }

                // Obtain 2 call contracts with different strike price to form the call spread
                var longCall = callContracts.First();
                var shortCall = callContracts.First(contract => contract.Strike > longCall.Strike);

                // Use all the buying power, but need to ensure the order size of the long and short call are the
same
                var quantity = new[] {
                    CalculateOrderQuantity(shortCall.Symbol, -1m),
                    CalculateOrderQuantity(longCall.Symbol, 1m) }
                    .Min(x=> Math.Abs(x));
                if (quantity == 0) return;

                // Bull call spread consists of long a lower-strike call and short a higher-strike call
                MarketOrder(shortCall.Symbol, -quantity);
                MarketOrder(longCall.Symbol, quantity);

                var expectedMarginUsage = Math.Max((longCall.Strike - shortCall.Strike) *
Securities[longCall.Symbol].SymbolProperties.ContractMultiplier * quantity, 0);
                if (expectedMarginUsage != Portfolio.TotalMarginUsed)
                {
                    throw new Exception("Unexpect margin used!");
                }
            }
        }
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            if (security.Type == SecurityType.IndexOption)
            {
                // Historical data
                var history = History(security.Symbol, 10, Resolution.Minute);
            }
        }
    }
}
```

Framework Algorithm Example

The following example algorithm buys the VIX Index Option Bull Call Spread with the furthest expiration:

C#

```
public class IndexOptionsDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(200000);
        UniverseSettings.Asynchronous = true;
        SetUniverseSelection(new VIXIndexOptionUniverseSelectionModel());

        SetAlpha(new BullCallSpreadAlphaModel());

        SetPortfolioConstruction(new SingleSharePortfolioConstructionModel());
    }
}

public class VIXIndexOptionUniverseSelectionModel : OptionUniverseSelectionModel
{
    // Only interest in VIX options
    public VIXIndexOptionUniverseSelectionModel()
        : base(TimeSpan.FromDays(1),
            _ => new[] {QuantConnect.Symbol.Create("VIX", SecurityType.IndexOption, Market.USA)})
    {
    }

    protected override OptionFilterUniverse Filter(OptionFilterUniverse optionFilterUniverse)
    {
        // Filter to get ATM calls expiring in 180 days to form the Bull Call Spread
        return optionFilterUniverse.CallsOnly().Strikes(-2, +2).Expiration(0, 180);
    }
}

public class BullCallSpreadAlphaModel : AlphaModel
{
    public BullCallSpreadAlphaModel()
    {
    }

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = Enumerable.Empty<Insight>();
        if (algorithm.Portfolio.Invested)
        {
            return insights;
        }

        foreach (var kvp in slice.OptionChains)
        {
            var canoncialSymbol = kvp.Key;
            var chain = kvp.Value;
            if (chain.IsNullOrEmpty())
            {
                return insights;
            }
            // Bull call spread legs are having the same expiry
            var expiry = chain.Max(contract => contract.Expiry);
            var callContracts = chain
                .Where(contract => contract.Expiry == expiry)
                .OrderBy(x => x.Strike)
                .ToList();

            // Need 2 contracts to form a call spread
            if (callContracts.Count < 2)
            {
                return insights;
            }

            // Obtain 2 call contracts with different strike price to form the call spread
            var longCall = callContracts.First();
            var shortCall = callContracts.First(contract => contract.Strike > longCall.Strike);

            // Bull call spread consists of long a lower-strike call and short a higher-strike call
            return Insight.Group(
                Insight.Price(longCall.Symbol, expiry.AddDays(1), InsightDirection.Up),
                Insight.Price(shortCall.Symbol, expiry.AddDays(1), InsightDirection.Down));
        }
        return insights;
```

```
            return insights;
        }
    }

    public class SingleSharePortfolioConstructionModel : PortfolioConstructionModel
    {
        public override IEnumerable<PortfolioTarget> CreateTargets(QCAlgorithm algorithm, Insight[] insights)
        {
            if (insights.Count() == 0)
            {
                return Enumerable.Empty<PortfolioTarget>();
            }
            // Only a single contract toi ensure both legs size is equal
            var quantity = insights
                .Select(i => Math.Abs(algorithm.CalculateOrderQuantity(i.Symbol, (decimal)i.Direction)))
                .Min();

            return insights.Select(i => new PortfolioTarget(i.Symbol, quantity * (decimal)i.Direction));
        }
    }
```

Data Point Attributes

The US Index Options dataset provides `TradeBar` , `QuoteBar` , and `OpenInterest` objects.

TradeBar Attributes

`TradeBar` objects have the following attributes:

QuoteBar Attributes

`QuoteBar` objects have the following attributes:

OpenInterest Attributes

`OpenInterest` objects have the following attributes:

Datasets

Morningstar

Morningstar was founded by Joe Mansueto in 1984 with the goal of empowering investors by connecting people to the investing information and tools they need. Morningstar provides access extensive line of products and services for individual investors, financial advisors, asset managers, and retirement plan providers. Morningstar provides data on approximately 525,000 investment offerings including stocks, mutual funds, and similar vehicles, along with real-time global market data on nearly 18 million equities, indexes, futures, options, commodities, and precious metals, in addition to foreign exchange and Treasury markets. Morningstar also offers investment management services through its investment advisory subsidiaries, with $244 billion in assets under advisement or management as of 2021.

US Fundamental Data

Morningstar

US Fundamental Data

Introduction

The US Fundamental Data by Morningstar tracks US Equity fundamentals. The data covers 8,000 US Equities, starts in January 1998, and is delivered on a daily frequency. The data does not include ETFs, ADRs, or OTC equities. This dataset is created by using a combination of string matching, Regular Expressions, and Machine Learning to gather the fundamental data published by companies.

For older symbols, the file date is approximated 45 days after the as of date. When a filing date is present on the Morningstar data, it is used. As we are a quant platform, all the data is loaded using "As Original Reported" figures. If there was a mistake reporting the figure, this will not be fixed later. The market typically responds quickly to these initially reported figures. Data is available for multiple periods depending on the property. Periods available include: 1 mo, 2 mo, 3 mo, 6 mo, 9 mo, 12 mo, 1 Yr, 2 Yr, 3 Yr, and 5 Yr. Morningstar symbols cover the NYSE, NASDAQ, AMEX, and BATS exchanges.

In live trading, Morningstar data is delivered to your algorithm at approximately 6 am each day. The majority of the fundamental data update occurs once per month. This includes updates to all of the key information for each security Morningstar supports. On top of this monthly update, there are daily updates of the financial ratios.

As Morningstar data arrives, it updates the master copy and is passed into your algorithm, similar to how TradeBars are fill-forwarded in your data feed. If there have been no updates for a day, you'll receive the same fundamental data.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

QuantConnect/LEAN combines the data of this dataset with US Coarse Universe data in runtime.

For more information about the US Fundamental Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Morningstar was founded by Joe Mansueto in 1984 with the goal of empowering investors by connecting people to the investing information and tools they need. Morningstar provides access extensive line of products and services for individual investors, financial advisors, asset managers, and retirement plan providers. Morningstar provides data on approximately 525,000 investment offerings including stocks, mutual funds, and similar vehicles, along with real-time global market data on nearly 18 million equities, indexes, futures, options, commodities, and precious metals, in addition to foreign exchange and Treasury markets. Morningstar also offers investment management services through its investment advisory subsidiaries, with $244 billion in assets under advisement or management as of 2021.

Getting Started

The following snippets demonstrate how to request data from the US Fundamental dataset:

- Direct Access using the Security object

```C#
var equity = AddEquity("IBM");
var ibmFundamental = equity.Fundamentals;
```

- Direct Access using the Symbol object

```
C#
var ibm = QuantConnect.Symbol.Create("IBM", SecurityType.Equity, Market.USA);
var ibmFundamental = Fundamentals(ibm);
```

- Universe Selection

```
C#
public override void Initialize()
{
    _universe = AddUniverseSelection(new FundamentalUniverseSelectionModel(FundamentalFilterFunction));
}

public override List<Symbol> FundamentalFilterFunction(List<Fundamental> fundamental)
{
    return (from f in fundamental
            where f.HasFundamentalData && f.Price > 10 && !Double.IsNaN(f.ValuationRatios.PERatio)
            orderby f.ValuationRatios.PERatio
            select f.Symbol).Take(10);
}
```

Data Summary

The following table describes the dataset properties:

Property	**Value**
Start Date	January 1998
Asset Coverage	8,000 US Equities *
Corporate Indicators / Tracked Fields	900 Fields
Data Density	Sparse
Resolution	Daily
Timezone	New York

* The data does not include ETFs, ADRs, or OTC equities.

Requesting Data

You don't need any special code to request US Fundamental Data. You can access the current and historical fundamental data for any of the US Equities that this dataset includes.

```C#
public class MorningStarDataAlgorithm : QCAlgorithm
{
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2021, 1, 1);
        SetEndDate(2021, 7, 1);
        SetCash(100000);
        UniverseSettings.Asynchronous = true;

        // Option 1: Subscribe to individual US Equity assets
        AddEquity("IBM");

        // Option 2A: Create a fundamental universe (classic version)
        _universe = AddUniverse(FundamentalFilterFunction);

        // Option 2B: Create a fundamental universe (framework version)
        AddUniverseSelection(new FundamentalUniverseSelectionModel(FundamentalFilterFunction));
    }
}
```

For more information about universe settings, see Settings .

Accessing Data

If you add a fundamental universe to your algorithm, you can access fundamental data in the universe selection function.

```C#
public IEnumerable<Symbol> FundamentalFunction(IEnumerable<Fundamental> fundamental)
{
    return fundamental.Where(f => f.Price > 10 && f.ValuationRatios.PERatios > 30).Select(f => f.Symbol);
}
```

To get fundamental data for Equities in your algorithm, use the `Fundamentals` property of the `Equity` objects. The fundamental data represent the corporate fundamentals for the current algorithm time.

```C#
var fundamentals = Securities[symbol].Fundamentals;
```

To get fundamental data for Equities, regardless of whether or not you have subscribed to them in your algorithm, call the `Fundamentals` method. If you pass one `Symbol` , the method returns a `Fundamental` object. If you pass a list of `Symbol` objects, the method returns a list of `Fundamental` objects.

```C#
// Single asset
var ibm = QuantConnect.Symbol.Create("IBM", SecurityType.Equity, Market.USA);
var ibmFundamental = Fundamentals(ibm);

// Multiple assets
var nb = QuantConnect.Symbol.Create("NB", SecurityType.Equity, Market.USA);
var fundamentals = Fundamentals(new List<Symbol>{ nb, ibm }).ToList();
```

Historical Data

You can get historical fundamental data in an algorithm and in the Research Environment.

Historical Data in Algorithms

To get historical fundamental data in an algorithm, call the `History` method with `Fundamental` type and the Equity `Symbol` or the `Universe` object. If there is no data in the period you request, the history result is empty.

```C#
var ibm = QuantConnect.Symbol.Create("IBM", SecurityType.Equity, Market.USA);

// Fundamental objects
var fundamentalHistory = History<Fundamental>(ibm, TimeSpan.FromDays(30));

// Fundamentals objects for all US Equities (including delisted companies)
var fundamentalsHistory = History<Fundamentals>(TimeSpan.FromDays(30));

// Collection of Fundamental objects for all US Equities (including delisted companies)
var collectionHistory = History(_universe, 30, Resolution.Daily);
foreach (var fundamental in collectionHistory)
{
    // Cast to Fundamental is required
    var highestMarketCap = fundamental.OfType<Fundamental>().OrderByDescending(x => x.MarketCap).Take(5);
}
```

Historical Data in the Research Environment

To get historical data in the Research Environment, call any of the preceding methods or call the `UniverseHistory` method with the `Universe` object, a start date, and an end date. This method returns the filtered universe. Alternatively, call the `GetFundamental` method with the Equity `Symbol` , a `Fundamental` property, a start date, and an end date. If there is no data in the period you request, the history result is empty.

```C#
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var fundamentals in universeHistory)
{
    foreach (Fundamental fundamental in fundamentals)
    {
        Console.WriteLine($"{fundamental.Symbol} market cap at {fundamental.EndTime}: {fundamental.MarketCap}");
    }
}

var history = qb.GetFundamental(symbol, "ValuationRatios.PERatios", new DateTime(2021, 1, 1), new DateTime(2021, 7,
1));
```

For more information about historical US Equity fundamental data, see Equity Fundamental Data .

Example Applications

The US Fundamentals dataset enables you to design strategies harnessing fundamental data points. Examples include the following strategies:

- Ranking a universe of securities by a value factor like the Piotroski F-score and buying a subset of the universe with the best factor ranking
- Using the Morningstar asset classification to target specific industries or to ensure your strategy is diversified across several sectors
- Trading securities that recently performed an IPO

Classic Algorithm Example

The following example algorithm creates a dynamic universe of US Equities. The first stage filter selects the 100 most liquid US Equities and the second stage filter selects the 10 assets with the greatest PE ratio. The algorithm then forms an equal-weighted portfolio with the 10 assets in the universe.

C#

```
public class MorningStarDataAlgorithm : QCAlgorithm
{
    private int _universeSize = 10;
    private SecurityChanges _changes = SecurityChanges.None;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Requesting data
        UniverseSettings.Resolution = Resolution.Daily;
        // Add universe selection to filter with fundamental data
        AddUniverse(FundamentalSelectionFunction);
    }

    public IEnumerable<Symbol> FundamentalSelectionFunction(IEnumerable<Fundamental> fundamental)
    {
        // Make sure fundamental data and PE ratio available
        // To avoid penny stocks with high volatility, set price above $1
        // Filter the top 100 high dollar volume equities to ensure liquidity
        // Get the top 10 PE Ratio stocks to follow speculations of a large number of capital
        return fundamental
            .Where(x => x.HasFundamentalData && x.Price > 1 && !Double.IsNaN(x.ValuationRatios.PERatio))
            .OrderByDescending(x => x.DollarVolume)
            .Take(100)
            .OrderByDescending(x => x.ValuationRatios.PERatio)
            .Take(_universeSize)
            .Select(x => x.Symbol);
    }

    public override void OnData(Slice slice)
    {
        // if we have no changes, do nothing
        if (_changes == SecurityChanges.None || slice.Bars.Count == 0) return;

        // liquidate removed securities, as not being the most popular stocks anymore
        foreach (var security in _changes.RemovedSecurities)
        {
            if (security.Invested)
            {
                Liquidate(security.Symbol);
            }
        }

        // we want 1/N allocation in each security in our universe to evenly dissipate risk
        foreach (var security in _changes.AddedSecurities)
        {
            SetHoldings(security.Symbol, 1m / _universeSize);
        }

        _changes = SecurityChanges.None;
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        _changes = changes;
    }
}
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of US Equities. The first stage filter selects the 100 most liquid US Equities and the second stage filter selects the 10 assets with the greatest PE ratio. The algorithm then forms an equal-weighted portfolio with the 10 assets in the universe.

```C#
public class MorningStarDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Add universe selection to filter with fundamental data
        UniverseSettings.Resolution = Resolution.Daily;
        AddUniverseSelection(new CustomFundamentalUniverseSelectionModel());

        SetAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(1)));

        // we want 1/N allocation in each security in our universe to evenly dissipate risk
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());

        AddRiskManagement(new NullRiskManagementModel());

        SetExecution(new ImmediateExecutionModel());
    }
}

public class CustomFundamentalUniverseSelectionModel : FundamentalUniverseSelectionModel
{
    public override IEnumerable<Symbol> Select(QCAlgorithm algorithm, IEnumerable<Fundamental> fundamental)
    {
        return fundamental
            // Make sure fundamental data and PE ratio available
            // To avoid penny stocks with high volatility, set price above $1
            .Where(x => x.HasFundamentalData && x.Price > 1 && !Double.IsNaN(x.ValuationRatios.PERatio))
            // Filter the top 100 high dollar volume equities to ensure liquidity
            .OrderByDescending(x => x.DollarVolume)
            .Take(100)
            // Get the top 10 PE Ratio stocks to follow speculations of a large number of capital
            .OrderByDescending(x => x.ValuationRatios.PERatio)
            .Take(10)
            .Select(x => x.Symbol);
    }
}
```

Research Example

The following example selects the 100 most liquid US Equities and lists the 10 assets with the greatest PE ratio.:

```C#
var qb = new QuantBook();

// Add Fundamental Universe Selection
IEnumerable<Symbol> FundamentalSelectionFunction(IEnumerable<Fundamental> fundamentals)
{
    return fundamentals
        .Where(x => !Double.IsNaN(x.ValuationRatios.PERatio))
        .OrderByDescending(x => x.DollarVolume).Take(100)
        .OrderByDescending(x => x.ValuationRatios.PERatio).Take(10)
        .Select(x => x.Symbol);
}

var universe = qb.AddUniverse(FundamentalSelectionFunction);

// Historical Universe data
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var fundamentals in universeHistory)
{
    foreach (Fundamental fundamental in fundamentals)
    {
        Console.WriteLine($"{fundamental.Symbol} PE ratio at {fundamental.EndTime}:
{fundamental.ValuationRatios.PERatio}");
    }
}
```

Data Point Attributes

The US Fundamentals dataset provides `Fundamental` objects. To filter `Fundamental` objects, you can use the `MorningstarSectorCode` , `MorningstarIndustryGroupCode` , and `MorningstarIndustryCode` enumeration values.

Fundamental Attributes

`Fundamental` objects have the following attributes:

MorningstarSectorCode Enumeration

Sectors are large super categories of data. To access the sector of an Equity, use the `MorningstarSectorCode` property.

```C#
filteredFundamentals = fundamental.Where(x =>
    x.AssetClassification.MorningstarSectorCode == MorningstarSectorCode.Technology);
```

The `MorningstarSectorCode` enumeration has the following members:

MorningstarIndustryGroupCode Enumeration

Industry groups are clusters of related industries that tie together. To access the industry group of an Equity, use the `MorningstarIndustryGroupCode` property.

```C#
filteredFundamentals = fundamental.Where(x =>
    x.AssetClassification.MorningstarIndustryGroupCode == MorningstarIndustryGroupCode.Software);
```

The `MorningstarIndustryGroupCode` enumeration has the following members:

MorningstarIndustryCode Enumeration

Industries are the finest classification level available and are the individual industries according to the Morningstar classification system. To access the industry group of an Equity, use the `MorningstarIndustryCode` property.

```C#
filteredFundamentals = fundamental.Where(x =>
    x.AssetClassification.MorningstarIndustryCode == MorningstarIndustryCode.SoftwareApplication);
```

The `MorningstarIndustryCode` enumeration has the following members:

Exchange ID Values

The exchange ID represents the exchange that lists the Equity. To access the exchange ID of an Equity, use the `PrimaryExchangeID` property.

```C#
filteredFundamentals = fundamental.Where(x =>
    x.CompanyReference.PrimaryExchangeID == "NAS");
```

The exchanges are represented by the following string values:

String Representation	Exchange
NYS	New York Stock Exchange (NYSE)

NAS	NASDAQ
ASE	American Stock Exchange (AMEX)
TSE	Tokyo Stock Exchange
AMS	Amsterdam Internet Exchange
SGO	Santiago Stock Exchange
XMAD	Madrid Stock Exchange
ASX	Australian Securities Exchange
BVMF	B3 (stock exchange)
LON	London Stock Exchange
TKS	Istanbul Stock Exchange Settlement and Custody Bank
SHG	Shanghai Exchange
LIM	Lima Stock Exchange
FRA	Frankfurt Stock Exchange
JSE	Johannesburg Stock Exchange
MIL	Milan Stock Exchange
TAE	Tel Aviv Stock Exchange
STO	Stockholm Stock Exchange
ETR	Deutsche Boerse Xetra Core
PAR	Paris Stock Exchange
BUE	Buenos Aires Stock Exchange
KRX	Korea Exchange
SWX	SIX Swiss Exchange
PINX	Pink Sheets (OTC)
CSE	Canadian Securities Exchange
PHS	Philippine Stock Exchange
MEX	Mexican Stock Exchange
TAI	Taiwan Stock Exchange
IDX	Indonesia Stock Exchange

OSL	Oslo Stock Exchange
BOG	Colombia Stock Exchange
NSE	National Stock Exchange of India
HEL	Nasdaq Helsinki
MISX	Moscow Exchange
HKG	Hong Kong Stock Exchange
IST	Istanbul Stock Exchange
BOM	Bombay Stock Exchange
TSX	Toronto Stock Exchange
BRU	Brussels Stock Exchange
BATS	BATS Global Markets
ARCX	NYSE Arca
GREY	Grey Market (OTC)
DUS	Dusseldorf Stock Exchange
BER	Berlin Stock Exchange
ROCO	Taipei Exchange
CNQ	Canadian Trading and Quotation System Inc.
BSP	Bangko Sentral ng Pilipinas
NEOE	NEO Exchange

Datasets

TickData

TickData was founded by a futures broker and a programmer in 1984 as the first company in the world to offer historical tick-by-tick prices on the futures and index markets. TickData provides access to comprehensive and detailed market data and analytics products in the financial industry covering Equities, Futures, Options, cash FOREX, and Cash Indices.

International Futures

TickData

International Futures

Introduction

The International Futures dataset by TickData provides Futures data, including price, volume, open interest, and expiry. The data covers the 4 contracts, DAX, FESX, HSI, and NKD, starting in July 1998 and is delivered on any frequency from tick to daily. This dataset is created by TickData, which negotiates directly with exchanges for their official archive and partners with real-time data providers with direct exchange connections and multiple redundant ticker plants.

This dataset also depends on the US Futures Security Master because the US Futures Security Master dataset contains information to construct continuous Futures.

For more information about the International Futures dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

TickData was founded by a futures broker and a programmer in 1984 as the first company in the world to offer historical tick-by-tick prices on the futures and index markets. TickData provides access to comprehensive and detailed market data and analytics products in the financial industry covering Equities, Futures, Options, cash FOREX, and Cash Indices.

Getting Started

The following snippet demonstrates how to request data from the International Futures dataset:

```C#
var hsi= AddFuture(Futures.Indices.HangSeng, Resolution.Minute).Symbol           // "HSI";
var dax = AddFuture(Futures.Indices.Dax, Resolution.Minute).Symbol               // "DAX";
var fesx = AddFuture(Futures.Indices.EuroStoxx50, Resolution.Minute).Symbol      // "FESX";
var nkd = AddFuture(Futures.Indices.Nikkei225Dollar, Resolution.Minute).Symbol   // "NKD";
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	July 1998, see `Supported Assets` below for details
Coverage	4 Contracts
Data Density	Dense
Resolution	Minute, Hour, & Daily
Timezone	Various, refer to `Supported Assets` below
Market Hours	Regular and Extended

Volume Discrepancies

For Futures contracts trading in the US, for example, NKD, see US Futures .

Requesting Data

To add International Futures data to your algorithm, call the `AddFuture` method. Save a reference to the Future so you can access the data later in your algorithm.

```
C#
public class InternationalFuturesDataAlgorithm : QCAlgorithm
{
    private Symbol _futureSymbol;

    public override void Initialize()
    {
        SetStartDate(2013, 12, 20);
        SetEndDate(2014, 2, 20);
        SetCash(1000000);
        UniverseSettings.Asynchronous = true;
        // Request Hang Seng Index Futures data
        var future = AddFuture(Futures.Indices.HangSeng);
        // Set filter to obtain the contracts expire within 90 days
        future.SetFilter(0, 90);
        _futureSymbol = future.Symbol;
    }
}
```

For more information about creating Future subscriptions, see Requesting Data or Futures Universes .

Accessing Data

To get the current International Futures data, index the `FuturesChains` property of the current `Slice` with the canonical Futures `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your Future at every time step.

```
C#
public override void OnData(Slice slice)
{
    // Get future chain of the canonical symbol
    if (slice.FuturesChains.TryGetValue(_futureSymbol, out var chain))
    {
        // Iterate each contract in the future chain
        foreach (var contract in chain)
        {
            Log($"{contract.Symbol} price at {slice.Time}: {contract.LastPrice}");
        }
    }
}
```

You can also iterate through all of the `FuturesChain` objects in the current `Slice` .

```
C#
public override void OnData(Slice slice)
{
    // Iterate all future chains
    foreach (var kvp in slice.FuturesChains)
    {
        var canonicalSymbol = kvp.Key;
        var chain = kvp.Value;
        // Iterate each contract in the future chain
        foreach (var contract in chain)
        {
            Log($"{contract.Symbol} price at {slice.Time}: {contract.LastPrice}");
        }
    }
}
```

For more information about accessing Futures data, see Handling Data .

Historical Data

You can get historical International Futures data in an algorithm and the Research Environment.

Historical Data In Algorithms

To get historical International Futures data in an algorithm, call the `History` method with the canonical Futures `Symbol` or a Futures contract `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
// TradeBar objects
var contractHistoryTradeBars = History(contract.Symbol, 100, Resolution.Minute);
var continuousHistoryTradeBars = History(
    symbols: new[] {_futureSymbol},
    start: Time - TimeSpan.FromDays(15),
    end: Time,
    resolution: Resolution.Minute,
    fillForward: false,
    extendedMarketHours: false,
    dataMappingMode: DataMappingMode.OpenInterest,
    dataNormalizationMode: DataNormalizationMode.Raw,
    contractDepthOffset: 0);

// QuoteBar objects
var contractHistoryQuoteBars = History<QuoteBar>(contract.Symbol, 100, Resolution.Minute);
var continuousHistoryQuoteBars = History<QuoteBar>(_futureSymbol, 100, Resolution.Minute);

// Tick objects
var contractHistoryTicks = History<Tick>(contract.Symbol, TimeSpan.FromSeconds(10), Resolution.Tick);
var continuousHistoryTicks = History<Tick>(_futureSymbol, TimeSpan.FromSeconds(10), Resolution.Tick);
```

For more information about historical data in algorithms, see History Requests . For more information about the price adjustments for continuous contracts, see Continous Contracts .

Historical Data In Research

To get historical International Futures data in the Research Environment for an entire Futures chain, call the `FutureHistory` method with the canonical Future `Symbol` .

```C#
var qb = new QuantBook();
var future = qb.AddFuture(Futures.Indices.HangSeng);
future.SetFilter(0, 90);
var history = qb.FutureHistory(future.Symbol, new DateTime(2020, 6, 1), new DateTime(2020, 6, 5));

var contracts = history.SelectMany(x => x.OptionChains.SelectMany(y => y.Value.Contracts.Keys)).Distinct().ToList();
var expiries = contracts.Select(x => x.ID.Date).Distinct().ToList();
```

To get historical data for a single International Futures contract or the continuous Futures contract, call the `History` method like you would in an algorithm but on the `QuantBook` object. For more information about historical data in the Research Environment, see Futures .

Supported Assets

The following table shows the available Futures:

Ticker	Future	Start Date	Time Zone	Currency
HSI	Hang Seng Index Futures	Jan 2010	Asia/Hong Kong	HKD
DAX	German Index Futures	Jan 2010	Europe/Berlin	EUR
FESX	EURO STOXX 50 Index Futures	Jul 1998	Europe/Berlin	EUR
NKD	Nikkei 225 Index Futures	Jan 2007	America/Chicago	USD

Example Applications

The International Futures dataset enables you to design Futures strategies accurately. Examples include the following strategies:

- Buying the Futures contract with the most open interest to reduce slippage and market impact
- Trade speculation on an International Index
- Trading bull calendar spreads to reduce volatility and margin requirements

Classic Algorithm Example -- HSI Futures

The following example algorithm uses ZigZag to determine the trend of Hang Seng Index. To trade the Indices, we make use of the HSI Futures.

C#

```
public class InternationalFuturesDataAlgorithm : QCAlgorithm
{
    private Future _hsiFuture;
    private ZigZag _zz;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Set the time zone to HKT to make it more comparable with the exchange.
        SetTimeZone(TimeZones.HongKong);
        // Set the account currency as HKD to trade HSI Futures.
        SetAccountCurrency("HKD", 1000000);

        // Seed the last price of the contracts for filling.
        Settings.SeedInitialPrices = true;

        // Request HSI Futures to trade.
        // Note that we will trade the contract with the highest open interest for liquidity.
        _hsiFuture = AddFuture(
            Futures.Indices.HangSeng,
            extendedMarketHours: true,
            dataMappingMode: DataMappingMode.LastTradingDay,
            contractDepthOffset: 0
        );
        // Request the corresponding underlying index for feeding indicators for trade signal generation.
        var hsiIndex = AddIndex("HSI").Symbol;

        // Create a ZigZag indicator to trade Hang Seng Index price pivot points.
        _zz = ZZ(hsiIndex, 0.15m, 5, Resolution.Daily);
        // Warm up indicator for immediate readiness to trade.
        WarmUpIndicator(hsiIndex, _zz, Resolution.Daily);
    }

    public override void OnData(Slice slice)
    {
        // Only place trade if the Future contracts is in market opening hours to avoid stale fills.
        if (IsMarketOpen(_hsiFuture.Symbol) && _zz.IsReady)
        {
            var pivot = _zz.PivotType;
            // If the last pivot point is low, the current trend is increasing after this low point.
            if (pivot == PivotPointType.Low && !Portfolio[_hsiFuture.Symbol].IsLong)
            {
                SetHoldings(_hsiFuture.Mapped, 0.2m);
            }
            // If the last pivot point is high, the current trend decreases after this high point.
            else if (pivot == PivotPointType.High && !Portfolio[_hsiFuture.Symbol].IsShort)
            {
                SetHoldings(_hsiFuture.Mapped, -0.2m);
            }
        }

        // Handle rollover in case the current mapped contract changes.
        foreach (var (_, changedEvent) in slice.SymbolChangedEvents)
        {
            var oldSymbol = changedEvent.OldSymbol;
            var newSymbol = AddFutureContract(changedEvent.NewSymbol).Symbol;
            var quantity = Portfolio[oldSymbol].Quantity;
            // Rolling over: to liquidate any position of the old mapped contract and switch to the newly mapped
contract
            Liquidate(oldSymbol);
            if (quantity != 0)
            {
                MarketOrder(newSymbol, quantity);
            }
        }
    }
}
```

Classic Algorithm Example -- FESX Futures

The following example algorithm uses the MACD indicator to determine the trend of the EuroStoxx50 Index. To trade the Index, we make use of the FESX Futures.

```C#
public class InternationalFuturesDataAlgorithm : QCAlgorithm
{
    private Future _fesxFuture;
    private MovingAverageConvergenceDivergence _macd;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Set the time zone to Berlin to make it more comparable with the exchange.
        SetTimeZone(TimeZones.Berlin);
        // Set the account currency as EUR to trade FSX Futures.
        SetAccountCurrency("EUR", 1000000);
        // Seed the last price of the contracts for filling.
        Settings.SeedInitialPrices = true;

        // Request FESX Futures to trade.
        // Note that we will trade the contract with the highest open interest for liquidity.
        _fesxFuture = AddFuture(
            Futures.Indices.EuroStoxx50,
            extendedMarketHours: true,
            dataMappingMode: DataMappingMode.LastTradingDay,
            contractDepthOffset: 0
        );
        // Request the corresponding underlying Index for feeding indicator for trade signal generation.
        var fesxIndex = AddIndex("SX5E").Symbol;

        // Create a MACD indicator to trade EuroStoxx trend changes on the fast-slow term trend convergence.
        _macd = MACD(fesxIndex, 12, 26, 9, resolution: Resolution.Daily);
        // Warm up indicator for immediate readiness to trade.
        WarmUpIndicator(fesxIndex, _macd, Resolution.Daily);
    }

    public override void OnData(Slice slice)
    {
        // Only place trade if the Future contracts is in market opening hours to avoid stale fills.
        if (IsMarketOpen(_fesxFuture.Symbol) && _macd.IsReady)
        {
            // To identify a sensitive change in trend, use the change in the histogram value of the MACD.
            var histogram = _macd.Histogram;
            var histogramChg = histogram - histogram.Previous;
            // Positive histogram value increasing means the fast trend is rising faster than the slow trend,
indicating a strong uptrend.
            if (histogramChg > 0 && histogram > 0 && !Portfolio[_fesxFuture.Symbol].IsLong)
            {
                SetHoldings(_fesxFuture.Mapped, 0.2m);
            }
            // Negative histogram value decreasing means the slow trend is dropping faster than the fast trend,
indicating a strong downtrend.
            else if (histogramChg < 0 && histogram < 0 && !Portfolio[_fesxFuture.Symbol].IsShort)
            {
                SetHoldings(_fesxFuture.Mapped, -0.2m);
            }
        }

        // Handle rollover in case the current mapped contract changes.
        foreach (var (_, changedEvent) in slice.SymbolChangedEvents)
        {
            var oldSymbol = changedEvent.OldSymbol;
            var newSymbol = AddFutureContract(changedEvent.NewSymbol).Symbol;
            var quantity = Portfolio[oldSymbol].Quantity;
            // Rolling over: to liquidate any position of the old mapped contract and switch to the newly mapped
contract
            Liquidate(oldSymbol);
            if (quantity != 0)
            {
                MarketOrder(newSymbol, quantity);
            }
        }
    }
}
```

Data Point Attributes

The International Futures dataset provides `Future` , `FuturesChain` , and `OpenInterest` objects. To configure the continuous

Future settings, use the `DataNormalizationMode` and `DataMappingMode` enumerations.

Future Attributes

`Future` objects have the following attributes:

FuturesChain Attributes

`FuturesChain` objects have the following attributes:

OpenInterest Attributes

`OpenInterest` objects have the following attributes:

DataNormalizationMode Values

The `DataNormalizationMode` enumeration has the following values:

DataMappingMode Values

The `DataMappingMode` enumeration has the following values:

Datasets

CoinAPI

CoinAPI was founded by Artur Pietrzyk in 2016 with the goal of providing real-time and historical cryptocurrency market data, collected from hundreds of exchanges. CoinAPI provides access to Cryptocurrencies for traders, market makers, and developers building third-party applications.

Binance Crypto Future Price Data

Binance Crypto Price Data

Binance US Crypto Price Data

Bitfinex Crypto Price Data

Bybit Crypto Future Price Data

Bybit Crypto Price Data

Coinbase Crypto Price Data

Kraken Crypto Price Data

CoinAPI

Binance Crypto Future Price Data

Introduction

The Binance Crypto Future Price Data by CoinAPI is for Crypto-currency futures price and volume data points. The data covers 718 Cryptocurrency pairs, starts in August 2020, and is delivered on any frequency from tick to daily. This dataset is created by monitoring the trading activity on Binance.

The Binance Crypto Future Margin Rate Data dataset provides margin interest data to model margin costs.

For more information about the Binance Crypto Future Price Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

CoinAPI was founded by Artur Pietrzyk in 2016 with the goal of providing real-time and historical cryptocurrency market data, collected from hundreds of exchanges. CoinAPI provides access to Cryptocurrencies for traders, market makers, and developers building third-party applications.

Getting Started

The following snippet demonstrates how to request data from the Binance Crypto Future Price dataset:

```C#
private Symbol _cryptoFutureSymbol;

public override void Initialize
{
    SetBrokerageModel(BrokerageName.BinanceFutures, AccountType.Margin);
    //SetBrokerageModel(BrokerageName.BinanceCoinFutures, AccountType.Margin);

    // perpetual futures does not have a filter function
    _cryptoFutureSymbol = AddCryptoFuture("BTCBUSD", Resolution.Minute).Symbol;
}
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	August 2020
Asset Coverage	718 Crypto Futures Pairs
Data Density	Dense
Resolution	Tick, Second, Minute, Hourly, & Daily
Timezone	UTC
Market Hours	Always Open

Requesting Data

To add Binance Crypto Future Price data to your algorithm, call the `AddCryptoFuture` method. Save a reference to the Crypto Future `Symbol` so you can access the data later in your algorithm.

```
C#
public class CoinAPIDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol ;

    public override void Initialize()
    {
        SetStartDate(2020, 6, 1);
        SetEndDate(2021, 6, 1);

        // Set Account Currency to Binance Stable Coin for USD
        SetAccountCurrency("BUSD");
        SetCash(100000);

        SetBrokerageModel(BrokerageName.BinanceFutures, AccountType.Margin);
        SetBrokerageModel(BrokerageName.BinanceCoinFutures, AccountType.Margin);

        var cryptoFuture = AddCryptoFuture("BTCBUSD", Resolution.Minute);
        // perpetual futures does not have a filter function
        _symbol = cryptoFuture.Symbol;
    }
}
```

For more information about creating Crypto Future subscriptions, see Requesting Data .

Accessing Data

To get the current Binance Crypto Future Price data, index the `Bars` , `QuoteBars` , or `Ticks` properties of the current `Slice` with the Crypto Future `Symbol` . `Slice` objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.Bars.ContainsKey(_symbol))
    {
        var tradeBar = slice.Bars[_symbol];
        Log($"{_symbol} price at {slice.Time}: {tradeBar.Close}");
    }

    if (slice.QuoteBars.ContainsKey(_symbol))
    {
        var quoteBar = slice.QuoteBars[_symbol];
        Log($"{_symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    if (slice.Ticks.ContainsKey(_symbol))
    {
        var ticks = slice.Ticks[_symbol];
        foreach (var tick in ticks)
        {
            Log($"{_symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

You can also iterate through all of the data objects in the current `Slice` .

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Bars)
    {
        var symbol = kvp.Key;
        var tradeBar = kvp.Value;
        Log($"{symbol} price at {slice.Time}: {tradeBar.Close}");
    }

    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        Log($"{symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    foreach (var kvp in slice.Ticks)
    {
        var symbol = kvp.Key;
        var ticks = kvp.Value;
        foreach (var tick in ticks)
        {
            Log($"{symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

For more information about accessing Crypto Future data, see Handling Data .

Historical Data

To get historical Binance Crypto Future Price data, call the `History` method with the Crypto Future `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
// TradeBar objects
var historyTradeBars = History(_symbol, 100, Resolution.Daily);

// QuoteBar objects
var historyQuoteBars = History<QuoteBar>(_symbol, 100, Resolution.Minute);

// Tick objects
var historyTicks = History<Tick>(_symbol, TimeSpan.FromSeconds(10), Resolution.Tick);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To unsubscribe from a Crypto Future contract that you added with the `AddCryptoFuture` method, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_symbol);
```

The `RemoveSecurity` method cancels your open orders for the security and liquidates your Crypto Future holdings.

Supported Assets

The following table shows the available Crypto Future pairs:

Pairs Available (718)

0GUSDT	1000000BOBUSDT	1000000MOGUSDT	1000BONKUSDC	1000BONKUSDT	1000CATUSDT
1000CHEEMSUSDT	1000FLOKIUSDT	1000LUNCBUSD	1000LUNCUSDT	1000PEPEUSDC	1000PEPEUSDT
1000RATSUSDT	1000SATSUSDT	1000SHIBBUSD	1000SHIBUSDC	1000SHIBUSDT	1000WHYUSDT
1000XECUSDT	1000XUSDT	1INCHUSDT	1MBABYDOGEUSDT	2ZUSDT	4USDT
A2ZUSDT	AAVEUSD	AAVEUSDC	AAVEUSDT	ACEUSDT	ACHUSDT
ACTUSDT	ACXUSDT	ADABUSD	ADAUSD	ADAUSDC	ADAUSDT
AERGOUSDT	AEROUSDT	AEVOUSDT	AGIXBUSD	AGIXUSDT	AGLDUSDT
AGTUSDT	AI16ZUSDT	AIAUSDT	AINUSDT	AIOTUSDT	AIOUSDT
AIUSDT	AIXBTUSDT	AKEUSDT	AKTUSDT	ALCHUSDT	ALGOUSD
ALGOUSDT	ALICEUSDT	ALLUSDT	ALPACAUSDT	ALPHAUSDT	ALPINEUSDT
ALTUSDT	AMBBUSD	AMBUSDT	ANCBUSD	ANIMEUSDT	ANKRUSDT
ANTUSDT	APEBUSD	APEUSD	APEUSDT	API3USDT	APTBUSD
APTUSD	APTUSDT	ARBUSDC	ARBUSDT	ARCUSDT	ARIAUSDT
ARKMUSDT	ARKUSDT	ARPAUSDT	ARUSDT	ASRUSDT	ASTERUSDT
ASTRUSDT	ATAUSDT	ATHUSDT	ATOMUSD	ATOMUSDT	AUCTIONBUSD
AUCTIONUSDT	AUDIOUSDT	AUSDT	AVAAIUSDT	AVAUSDT	AVAXBUSD
AVAXUSD	AVAXUSDC	AVAXUSDT	AVNTUSDT	AWEUSDT	AXLUSDT
AXSUSD	AXSUSDT	B2USDT	B3USDT	BABYUSDT	BADGERUSDT
BAKEUSDT	BALUSDT	BANANAS31USDT	BANANAUSDT	BANDUSDT	BANKUSDT
BANUSDT	BARDUSDT	BASUSDT	BATUSDT	BBUSDT	BCHUSD
BCHUSDC	BCHUSDT	BDXNUSDT	BEAMXUSDT	BELUSDT	BERAUSDT
BICOUSDT	BIDUSDT	BIGTIMEUSDT	BIOUSDC	BIOUSDT	BLESSUSDT
BLUAIUSDT	BLUEBIRDUSDT	BLURUSDT	BLZUSDT	BMTUSDT	BNBBUSD
BNBUSD	BNBUSDC	BNBUSDT	BNTUSDT	BNXUSDT	BOMEUSDC
BOMEUSDT	BONDUSDT	BRETTUSDT	BROCCOLI714USDT	BROCCOLIF3BUSDT	BRUSDT
BSVUSDT	BSWUSDT	BTCBUSD	BTCDOMUSDT	BTCSTUSDT	BTCUSD
BTCUSDC	BTCUSDT	BTRUSDT	BTSUSDT	BULLAUSDT	BUSDT
C98USDT	CAKEUSDT	CARVUSDT	CATIUSDT	CELOUSDT	CELRUSDT

CETUSUSDT	CFXUSDT	CGPTUSDT	CHESSUSDT	CHILLGUYUSDT	CHRUSDT
CHZUSD	CHZUSDT	CKBUSDT	CLOUSDT	COAIUSDT	COCOSUSDT
COMBOUSDT	COMPUSDT	COOKIEUSDT	COSUSDT	COTIUSDT	COWUSDT
CROSSUSDT	CRVUSDC	CRVUSDT	CTKUSDT	CTSIUSDT	CUDISUSDT
CUSDT	CVCUSDT	CVXBUSD	CVXUSDT	CYBERUSDT	DAMUSDT
DARUSDT	DASHUSDT	DEEPUSDT	DEFIUSDT	DEGENUSDT	DEGOUSDT
DENTUSDT	DEXEUSDT	DFUSDT	DGBUSDT	DIAUSDT	DMCUSDT
DODOBUSD	DODOXUSDT	DOGEBUSD	DOGEUSD	DOGEUSDC	DOGEUSDT
DOGSUSD	DOGSUSDT	DOLOUSDT	DOODUSDT	DOTBUSD	DOTUSD
DOTUSDT	DRIFTUSDT	DUSDT	DUSKUSDT	DYDXUSDT	DYMUSDT
EDENUSDT	EDUUSDT	EGLDUSD	EGLDUSDT	EIGENUSDT	ENAUSDC
ENAUSDT	ENJUSDT	ENSOUSDT	ENSUSD	ENSUSDT	EOSUSD
EOSUSDT	EPICUSDT	EPTUSDT	ERAUSDT	ESPORTSUSDT	ETCBUSD
ETCUSD	ETCUSDT	ETHBTC	ETHBUSD	ETHFIUSDC	ETHFIUSDT
ETHUSD	ETHUSDC	ETHUSDT	ETHWUSDT	EULUSDT	EVAAUSDT
FARTCOINUSDT	FETUSDT	FFUSDT	FHEUSDT	FIDAUSDT	FILBUSD
FILUSD	FILUSDC	FILUSDT	FIOUSDT	FISUSDT	FLMUSDT
FLOCKUSDT	FLOWUSDT	FLUIDUSDT	FLUXUSDT	FOOTBALLUSDT	FORMUSDT
FORTHUSDT	FRONTUSDT	FTMBUSD	FTMUSD	FTMUSDT	FTTBUSD
FTTUSDT	FUNUSDT	FUSDT	FXSUSDT	GALABUSD	GALAUSD
GALAUSDT	GALBUSD	GALUSDT	GASUSDT	GHSTUSDT	GIGGLEUSDT
GLMRUSDT	GLMUSDT	GMTBUSD	GMTUSD	GMTUSDT	GMXUSDT
GOATUSDT	GPSUSDT	GRASSUSDT	GRIFFAINUSDT	GRTUSDT	GTCUSDT
GUNUSDT	GUSDT	HAEDALUSDT	HANAUSDT	HBARUSDC	HBARUSDT
HEIUSDT	HEMIUSDT	HFTUSDT	HIFIUSDT	HIGHUSDT	HIPPOUSDT
HIVEUSDT	HMSTRUSDT	HNTUSDT	HOLOUSDT	HOMEUSDT	HOOKUSDT
HOTUSDT	HUMAUSDT	HUSDT	HYPERUSDT	HYPEUSDT	ICNTUSDT
ICPBUSD	ICPUSDT	ICXUSD	ICXUSDT	IDEXUSDT	IDOLUSDT

IDUSDT	ILVUSDT	IMXUSDT	INITUSDT	INJUSDT	INUSDT
IOSTUSDT	IOTAUSDT	IOTXUSDT	IOUSDT	IPUSDC	IPUSDT
JASMYUSDT	JELLYJELLYUSDT	JOEUSDT	JSTUSDT	JTOUSDT	JUPUSDT
KAIAUSDT	KAITOUSDC	KAITOUSDT	KASUSDT	KAVAUSDT	KDAUSDT
KERNELUSDT	KEYUSDT	KGENUSDT	KLAYUSDT	KMNOUSDT	KNCUSD
KNCUSDT	KOMAUSDT	KSMUSDT	LABUSDT	LAUSDT	LAYERUSDT
LDOBUSD	LDOUSDT	LEVERBUSD	LEVERUSDT	LIGHTUSDT	LINAUSDT
LINEAUSDT	LINKBUSD	LINKUSD	LINKUSDC	LINKUSDT	LISTAUSDT
LITUSDT	LOKAUSDT	LOOMUSDT	LPTUSDT	LQTYUSDT	LRCUSDT
LSKUSDT	LTCBUSD	LTCUSD	LTCUSDC	LTCUSDT	LUMIAUSDT
LUNA2BUSD	LUNA2USDT	LYNUSDT	MAGICUSDT	MANAUSD	MANAUSDT
MANTAUSDT	MASKUSDT	MATICBUSD	MATICUSD	MATICUSDC	MATICUSDT
MAVIAUSDT	MAVUSDT	MBLUSDT	MBOXUSDT	MDTUSDT	MELANIAUSDT
MEMEFIUSDT	MEMEUSDT	MERLUSDT	METISUSDT	METUSDT	MEUSDT
MEWUSDT	MILKUSDT	MINAUSDT	MIRAUSDT	MITOUSDT	MKRUSDT
MLNUSDT	MOCAUSDT	MONUSDT	MOODENGUSDT	MORPHOUSDT	MOVEUSDT
MOVRUSDT	MTLUSDT	MUBARAKUSDT	MUSDT	MYROUSDT	MYXUSDT
NAORISUSDT	NEARBUSD	NEARUSD	NEARUSDC	NEARUSDT	NEIROETHUSDT
NEIROUSDT	NEOUSDC	NEOUSDT	NEWTUSDT	NFPUSDT	NILUSDT
NKNUSDT	NMRUSDT	NOMUSDT	NOTUSDT	NTRNUSDT	NULSUSDT
NXPCUSDT	OBOLUSDT	OCEANUSDT	OGNUSDT	OGUSDT	OLUSDT
OMGUSDT	OMNIUSDT	OMUSDT	ONDOUSDT	ONEUSDT	ONGUSDT
ONTUSDT	OPENUSDT	OPUSD	OPUSDT	ORBSUSDT	ORCAUSDT
ORDERUSDT	ORDIUSDC	ORDIUSDT	OXTUSDT	PARTIUSDT	PAXGUSDT
PENDLEUSDT	PENGUUSDC	PENGUUSDT	PEOPLEUSDT	PERPUSDT	PHAUSDT
PHBBUSD	PHBUSDT	PIPPINUSDT	PIXELUSDT	PLAYUSDT	PLUMEUSDT
PNUTUSDC	PNUTUSDT	POLUSDT	POLYXUSDT	PONKEUSDT	POPCATUSDT
PORT3USDT	PORTALUSDT	POWRUSDT	PROMPTUSDT	PROMUSDT	PROVEUSDT

PTBUSDT	PUFFERUSDT	PUMPBTCUSDT	PUMPUSDT	PUNDIXUSDT	PYTHUSDT
QNTUSDT	QTUMUSDT	QUICKUSDT	QUSDT	RADUSDT	RAREUSDT
RAYSOLUSDT	RAYUSDT	RDNTUSDT	RECALLUSDT	REDUSDT	REEFUSDT
REIUSDT	RENDERUSDT	RENUSDT	RESOLVUSDT	REZUSDT	RIFUSDT
RIVERUSDT	RLCUSDT	RNDRUSDT	RONINUSDT	ROSEUSD	ROSEUSDT
RPLUSDT	RSRUSDT	RUNEUSD	RUNEUSDT	RVNUSDT	RVVUSDT
SAFEUSDT	SAGAUSDT	SAHARAUSDT	SANDBUSD	SANDUSD	SANDUSDT
SANTOSUSDT	SAPIENUSDT	SCRTUSDT	SCRUSDT	SCUSDT	SEIUSDT
SFPUSDT	SHELLUSDT	SIGNUSDT	SIRENUSDT	SKATEUSDT	SKLUSDT
SKYAIUSDT	SKYUSDT	SLERFUSDT	SLPUSDT	SNTUSDT	SNXUSDT
SOLBUSD	SOLUSD	SOLUSDC	SOLUSDT	SOLVUSDT	SOMIUSDT
SONICUSDT	SOONUSDT	SOPHUSDT	SPELLUSDT	SPKUSDT	SPXUSDT
SQDUSDT	SRMUSDT	SSVUSDT	STBLUSDT	STEEMUSDT	STGUSDT
STMXUSDT	STORJUSDT	STOUSDT	STPTUSDT	STRAXUSDT	STRKUSDT
STXUSDT	SUIUSD	SUIUSDC	SUIUSDT	SUNUSDT	SUPERUSDT
SUSDT	SUSHIUSDT	SWARMSUSDT	SWELLUSDT	SXPUSDT	SXTUSDT
SYNUSDT	SYRUPUSDT	SYSUSDT	TACUSDT	TAGUSDT	TAIKOUSDT
TAKEUSDT	TANSSIUSDT	TAOUSDT	TAUSDT	THETAUSD	THETAUSDT
THEUSDT	TIAUSDC	TIAUSDT	TLMBUSD	TLMUSDT	TNSRUSDT
TOKENUSDT	TOMOUSDT	TONUSDT	TOSHIUSDT	TOWNSUSDT	TRADOORUSDT
TRBUSDT	TREEUSDT	TROYUSDT	TRUMPUSDC	TRUMPUSDT	TRUTHUSDT
TRUUSDT	TRXBUSD	TRXUSD	TRXUSDT	TSTUSDT	TURBOUSDT
TUSDT	TUTUSDT	TWTUSDT	UBUSDT	UMAUSDT	UNFIUSDT
UNIBUSD	UNIUSD	UNIUSDC	UNIUSDT	USDCUSDT	USELESSUSDT
USTCUSDT	USUALUSDT	UXLINKUSDT	VANAUSDT	VANRYUSDT	VELODROMEUSDT
VELVETUSDT	VETUSD	VETUSDT	VFYUSDT	VICUSDT	VIDTUSDT
VINEUSDT	VIRTUALUSDT	VOXELUSDT	VTHOUSDT	VVVUSDT	WALUSDT
WAVESBUSD	WAVESUSDT	WAXPUSDT	WCTUSDT	WIFUSD	WIFUSDC

WIFUSDT	WLDUSD	WLDUSDC	WLDUSDT	WLFIUSDC	WLFIUSDT
WOOUSDT	WUSDT	XAIUSDT	XANUSDT	XCNUSDT	XEMUSDT
XLMUSD	XLMUSDT	XMRUSD	XMRUSDT	XNYUSDT	XPINUSDT
XPLUSDT	XRPBUSD	XRPUSD	XRPUSDC	XRPUSDT	XTZUSD
XTZUSDT	XVGUSDT	XVSUSDT	YALAUSDT	YBUSDT	YFIUSDT
YGGUSDT	ZBTUSDT	ZECUSDT	ZENUSDT	ZEREBROUSDT	ZETAUSDT
ZILUSD	ZILUSDT	ZKCUSDT	ZKJUSDT	ZKUSDT	ZORAUSDT
ZRCUSDT	ZROUSDT	ZRXUSDT	[illegible]USDT		

Example Applications

The Binance Crypto Future Price dataset enables you to accurately design strategies for Cryptocurrencies with term structure. Examples include the following strategies:

- Horizontal/Diagonal arbitrage with the underlying cryptocurrencies
- Trade Contango/Backwardation predictions
- Hedge for illiquid cryptocurrencies

Classic Algorithm Example

The following example algorithm buys BTCUSDT perpetual future contract if the last day's close price was close to ask close price than bid close price, sells short of that in opposite, through the Binance exchange:

```
from AlgorithmImports import * class BinanceCryptoFutureDataAlgorithm(QCAlgorithm): def initialize(se
lf) -> None: self.set_start_date(2024, 9, 1) self.set_end_date(2024, 12, 31) # Set the account curren
cy to USDT, since you can't trade with # USD and the algorithm doesn't automatically convert USD & US
DT # holdings. self.set_account_currency("USDT", 100000) # Binance Futures Exchange accepts both Cash
and Margin account # types. Select the one you need for the best reality modeling. self.set_brokerage
_model(BrokerageName.BINANCE_FUTURES, AccountType.MARGIN) # Add BTCUSDT data from the Binance Future
exchange. # Perpetual futures does not have a filter function. self._btc = self.add_crypto_future("BT
CUSDT", Resolution.DAILY) # Historical data history = self.history(self._btc, 10, Resolution.DAILY) d
ef on_data(self, slice: Slice) -> None: # Trade only based on updated price data. if self._btc.symbol
not in slice.bars or self._btc.symbol not in slice.quote_bars: return # Scalp-trade the bid-ask sprea
d based on the supply-demand strength if self._btc.price - self._btc.bid_price > self._btc.ask_price
- self._btc.price: self.set_holdings(self._btc, -0.5) else: self.set_holdings(self._btc, 1)
```

```
public class BinanceCryptoFutureDataAlgorithm : QCAlgorithm { public CryptoFuture _btc; public overri
de void Initialize() { SetStartDate(2024, 9, 1); SetEndDate(2024, 12, 31); // Set the account currenc
y to USDT, since you can't trade with // USD and the algorithm doesn't automatically convert USD & US
DT // holdings. SetAccountCurrency("USDT", 100000); // Binance Futures Exchange accepts both Cash and
Margin account // types. Select the one you need for the best reality modeling. SetBrokerageModel(Bro
kerageName.BinanceFutures, AccountType.Margin); // Add BTCUSDT data from the Binance Future exchange.
// Perpetual futures does not have a filter function. _btc = AddCryptoFuture("BTCUSDT", Resolution.Da
ily); // Historical data var history = History(_btc, 10, Resolution.Daily); } public override void On
Data(Slice slice) { // Trade only based on updated price data. if (!slice.Bars.ContainsKey(_btc) || !
slice.QuoteBars.ContainsKey(_btc)) { return; } // Scalp-trade the bid-ask spread based on the supply-
demand strength if (_btc.Price - _btc.BidPrice > _btc.AskPrice - _btc.Price) { SetHoldings(_btc, -0.5
m); } else { SetHoldings(_btc, 1m); } } }
```

Framework Algorithm Example

The following example algorithm hold a long BTCUSDT Future portfolio if the last day's close price was close to ask close price than bid close price, while hold short of that in opposite, through the Binance exchange using the algorithm framework implementation:

```
from AlgorithmImports import * class BinanceCryptoFutureDataAlgorithm(QCAlgorithm): def initialize(se
lf) -> None: self.set_start_date(2024, 9, 1) self.set_end_date(2024, 12, 31) # Set the account curren
cy to USDT, since you can't trade with # USD and the algorithm doesn't automatically convert USD & US
DT # holdings. self.set_account_currency("USDT", 100000) # Binance Futures Exchange accepts both Cash
and Margin account types, select the one you need for the best reality modeling. self.set_brokerage_m
odel(BrokerageName.BINANCE_FUTURES, AccountType.MARGIN) self.universe_settings.resolution = Resolutio
n.DAILY self.universe_settings.leverage = 2 # We only trade on BTCUSDT Future in Binance Future excha
nge symbols = Symbol.create("BTCUSDT", SecurityType.CRYPTO_FUTURE, Market.BINANCE) self.add_universe_
selection(ManualUniverseSelectionModel(symbols)) # Custom alpha model to emit insights based on the C
rypto Future price data self.add_alpha(CryptoFutureAlphaModel()) # Equally invest to evenly dissipate
the capital concentration risk of inidividual crypto pair self.set_portfolio_construction(EqualWeight
ingPortfolioConstructionModel()) self.set_execution(ImmediateExecutionModel()) class CryptoFutureAlph
aModel(AlphaModel): def __init__(self) -> None: self.symbols = [] def update(self, algorithm: QCAlgor
ithm, slice: Slice) -> List[Insight]: insights = [] for symbol in self.symbols: # Trade only based on
updated price data if not slice.bars.contains_key(symbol) or not slice.quote_bars.contains_key(symbol
): continue quote = slice.quote_bars[symbol] price = slice.bars[symbol].price # Scalp-trade the bid-a
sk spread based on the supply-demand strength if price - quote.bid.close > quote.ask.close - price: d
irection = InsightDirection.DOWN else: direction = InsightDirection.UP insights.append(Insight.price(
symbol, timedelta(1), direction)) return insights def on_securities_changed(self, algorithm: QCAlgori
thm, changes: SecurityChanges) -> None: for security in changes.added_securities: symbol = security.s
ymbol self.symbols.append(symbol) # Historical data history = algorithm.history(symbol, 10, Resolutio
n.DAILY) for security in changes.removed_securities: symbol = security.symbol if symbol in self.symbo
ls: self.symbols.remove(symbol)
```

```
public class BinanceCryptoFutureDataAlgorithm : QCAlgorithm { public Symbol _symbol; public override
void Initialize() { SetStartDate(2024, 9, 1); SetEndDate(2024, 12, 31); // Set the account currency t
o USDT, since you can't trade with // USD and the algorithm doesn't automatically convert USD & USDT
// holdings. SetAccountCurrency("USDT", 100000); // Binance Futures Exchange accepts both Cash and Ma
rgin account types, select the one you need for the best reality modeling. SetBrokerageModel(Brokerag
eName.BinanceFutures, AccountType.Margin); UniverseSettings.Resolution = Resolution.Daily; UniverseSe
ttings.Leverage = 2; // We only trade on BTCUSDT Future in Binance Future exchange var symbols = new
List<Symbol> { QuantConnect.Symbol.Create("BTCUSDT", SecurityType.CryptoFuture, Market.Binance) }; Ad
dUniverseSelection(new ManualUniverseSelectionModel(symbols)); // Custom alpha model to emit insights
based on the Crypto Future price data AddAlpha(new CryptoFutureAlphaModel()); // Equally invest to ev
enly dissipate the capital concentration risk of inidividual crypto pair SetPortfolioConstruction(new
EqualWeightingPortfolioConstructionModel()); SetExecution(new ImmediateExecutionModel()); } } public
class CryptoFutureAlphaModel : AlphaModel { private List<Symbol> _symbols = new(); public override IE
numerable<Insight> Update(QCAlgorithm algorithm, Slice slice) { var insights = new List<Insight>(); f
oreach (var symbol in _symbols) { // Trade only based on updated price data if (!slice.Bars.ContainsK
ey(symbol) || !slice.QuoteBars.TryGetValue(symbol, out var quote)) { continue; } var price = slice.Ba
rs[symbol].Price; // Scalp-trade the bid-ask spread based on the supply-demand strength InsightDirect
ion direction; if (price - quote.Bid.Close > quote.Ask.Close - price) { direction = InsightDirection.
Down; } else { direction = InsightDirection.Up; } insights.Add(Insight.Price(symbol, TimeSpan.FromDay
s(1), direction)); } return insights; } public override void OnSecuritiesChanged(QCAlgorithm algorith
m, SecurityChanges changes) { foreach (var security in changes.AddedSecurities) { var symbol = securi
ty.Symbol; _symbols.Add(symbol); // Historical data var history = algorithm.History(symbol, 10, Resol
ution.Daily); } foreach (var security in changes.RemovedSecurities) { _symbols.Remove(security.Symbol
); } } }
```

Data Point Attributes

The Binance Crypto Future Price dataset provides `TradeBar` , `QuoteBar` , and `Tick` objects.

TradeBar Attributes

`TradeBar` objects have the following attributes:

QuoteBar Attributes

`QuoteBar` objects have the following attributes:

Tick Attributes

`Tick` objects have the following attributes:

CoinAPI

Binance Crypto Price Data

Introduction

The Binance Crypto Price Data by CoinAPI is for Cryptocurrency price and volume data points. The data covers 3,320 Cryptocurrency pairs, starts in July 2017, and is delivered on any frequency from tick to daily. This dataset is created by monitoring the trading activity on Binance.

For more information about the Binance Crypto Price Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

CoinAPI was founded by Artur Pietrzyk in 2016 with the goal of providing real-time and historical cryptocurrency market data, collected from hundreds of exchanges. CoinAPI provides access to Cryptocurrencies for traders, market makers, and developers building third-party applications.

Getting Started

The following snippet demonstrates how to request data from the Binance Crypto Price dataset:

```C#
// Binance accepts both Cash and Margin account types only.
SetBrokerageModel(BrokerageName.Binance, AccountType.Cash);
//SetBrokerageModel(BrokerageName.Binance, AccountType.Margin);

_symbol = AddCrypto("BTCBUSD", Resolution.Minute, Market.Binance).Symbol;

_universe = AddUniverse(CryptoUniverse.Binance(UniverseSelectionFilter));
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	July 2017
Asset Coverage	3,320 Currency Pairs
Data Density	Dense
Resolution	Tick, Second, Minute, Hourly, & Daily
Timezone	UTC
Market Hours	Always Open

Requesting Data

To add Binance Crypto Price data to your algorithm, call the `AddCrypto` method. Save a reference to the Crypto `Symbol` so you can access the data later in your algorithm.

```C#
public class CoinAPIDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2020, 6, 1);
        SetEndDate(2021, 6, 1);

        // Set Account Currency to Binance Stable Coin for USD
        SetAccountCurrency("BUSD");
        SetCash(100000);

        // Binance accepts both Cash and Margin account types.
        SetBrokerageModel(BrokerageName.Binance, AccountType.Margin);

        _symbol = AddCrypto("BTCBUSD", Resolution.Minute, Market.Binance).Symbol;
    }
}
```

For more information about creating Crypto subscriptions, see Requesting Data .

Accessing Data

To get the current Binance Crypto Price data, index the `Bars` , `QuoteBars` , or `Ticks` properties of the current `Slice` with the Crypto `Symbol` . `Slice` objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```C#
public override void OnData(Slice slice)
{
    if (slice.Bars.ContainsKey(_symbol))
    {
        var tradeBar = slice.Bars[_symbol];
        Log($"{_symbol} price at {slice.Time}: {tradeBar.Close}");
    }

    if (slice.QuoteBars.ContainsKey(_symbol))
    {
        var quoteBar = slice.QuoteBars[_symbol];
        Log($"{_symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    if (slice.Ticks.ContainsKey(_symbol))
    {
        var ticks = slice.Ticks[_symbol];
        foreach (var tick in ticks)
        {
            Log($"{_symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

You can also iterate through all of the data objects in the current `Slice` .

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Bars)
    {
        var symbol = kvp.Key;
        var tradeBar = kvp.Value;
        Log($"{symbol} price at {slice.Time}: {tradeBar.Close}");
    }

    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        Log($"{symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    foreach (var kvp in slice.Ticks)
    {
        var symbol = kvp.Key;
        var ticks = kvp.Value;
        foreach (var tick in ticks)
        {
            Log($"{symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

For more information about accessing Crypto data, see Handling Data .

Historical Data

To get historical Binance Crypto Price data, call the `History` method with the Crypto `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
// TradeBar objects
var historyTradeBars = History(_symbol, 100, Resolution.Daily);

// QuoteBar objects
var historyQuoteBars = History<QuoteBar>(_symbol, 100, Resolution.Minute);

// Tick objects
var historyTicks = History<Tick>(_symbol, TimeSpan.FromSeconds(10), Resolution.Tick);
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of Binance Crypto pairs, call the `AddUniverse` method with a `CryptoUniverse` object. A Crypto universe uses a selection function to select Crypto pairs based on their OHLCV and dollar volume of the previous day as of midnight Coordinated Universal Time (UTC).

```
C#
using QuantConnect.Data.UniverseSelection;

public override void Initialize()
{
    UniverseSettings.Asynchronous = true;
    SetBrokerageModel(BrokerageName.Binance, AccountType.Margin);
    _universe = AddUniverse(CryptoUniverse.Binance(UniverseSelectionFilter));
}

private IEnumerable<Symbol> UniverseSelectionFilter(IEnumerable<CryptoUniverse> universeDay)
{
    return from c in universeDay
           where c.Volume >= 100m && c.VolumeInUsd > 10000m
           select c.Symbol;
}
```

For more information about universe settings, see Settings .

Universe History

You can get historical universe data in an algorithm and in the Research Environment.

Historical Universe Data in Algorithms

To get historical universe data in an algorithm, call the `History` method with the `Universe` object, and the lookback period. If there is no data in the period you request, the history result is empty.

```
C#
var history = History(_universe, 30, Resolution.Daily);
foreach (var universeDay in history)
{
    foreach (CryptoUniverse universeItem in universeDay)
    {
        Log($"{universeItem.Symbol} price at {universeItem.EndTime}: {universeItem.Close}");
    }
}
```

Historical Universe Data in Research

To get historical universe data in research, call the `UniverseHistory` method with the `Universe` object, and the lookback period. The `UniverseHistory` returns the filtered universe. If there is no data in the period you request, the history result is empty.

```
C#
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var universeDay in universeHistory)
{
    foreach (CryptoUniverse universeItem in universeDay)
    {
        Console.WriteLine($"{universeItem.Symbol} price at {universeItem.EndTime}: {universeItem.Close}");
    }
}
```

You can call the `History` method in Research.

Remove Subcriptions

To unsubscribe from a Crypto pair that you added with the `AddCrypto` method, call the `RemoveSecurity` method.

```
C#
RemoveSecurity(_symbol);
```

The `RemoveSecurity` method cancels your open orders for the security and liquidates your holdings in the virtual pair .

Supported Assets

The following table shows the available Cryptocurrency pairs:

Pairs Available (3320)					
0GBNB	0GFDUSD	0GTRY	0GUSDC	0GUSDT	1000CATBNB
1000CATFDUSD	1000CATTRY	1000CATUSDC	1000CATUSDT	1000CHEEMSUSDC	1000CHEEMSUSDT
1000SATSFDUSD	1000SATSTRY	1000SATSUSDC	1000SATSUSDT	1INCHBTC	1INCHBUSD
1INCHUSDT	1MBABYDOGEFDUSD	1MBABYDOGETRY	1MBABYDOGEUSDC	1MBABYDOGEUSDT	2ZBNB
2ZFDUSD	2ZTRY	2ZUSDC	2ZUSDT	A2ZUSDC	A2ZUSDT
AAVEBKRW	AAVEBNB	AAVEBRL	AAVEBTC	AAVEBUSD	AAVEETH
AAVEFDUSD	AAVETRY	AAVEUSDC	AAVEUSDT	ABTC	ACABTC
ACABUSD	ACATRY	ACAUSDT	ACEBNB	ACEBTC	ACEFDUSD
ACETRY	ACEUSDT	ACHBTC	ACHBUSD	ACHTRY	ACHUSDC
ACHUSDT	ACMBTC	ACMBUSD	ACMTRY	ACMUSDT	ACTBRL
ACTEUR	ACTFDUSD	ACTTRY	ACTUSDC	ACTUSDT	ACXFDUSD
ACXTRY	ACXUSDC	ACXUSDT	ADAAUD	ADABIDR	ADABKRW
ADABNB	ADABRL	ADABTC	ADABUSD	ADAETH	ADAEUR
ADAFDUSD	ADAGBP	ADAJPY	ADAPAX	ADARUB	ADATRY
ADATUSD	ADAUSDC	ADAUSDT	ADXBNB	ADXBTC	ADXBUSD
ADXETH	ADXUSDT	AEBNB	AEBTC	AEETH	AERGOBTC
AERGOBUSD	AERGOUSDT	AETH	AEURUSDT	AEVOBNB	AEVOBTC
AEVOFDUSD	AEVOTRY	AEVOUSDC	AEVOUSDT	AFDUSD	AGIBNB
AGIBTC	AGIETH	AGIXBTC	AGIXBUSD	AGIXFDUSD	AGIXTRY
AGIXUSDT	AGLDBNB	AGLDBTC	AGLDBUSD	AGLDUSDT	AIBNB
AIBTC	AIFDUSD	AIONBNB	AIONBTC	AIONBUSD	AIONETH
AIONUSDT	AITRY	AITUSD	AIUSDT	AIXBTTRY	AIXBTUSDC
AIXBTUSDT	AKROBTC	AKROBUSD	AKROUSDT	ALCXBTC	ALCXBUSD
ALCXUSDT	ALGOBIDR	ALGOBNB	ALGOBTC	ALGOBUSD	ALGOETH

ALGOFDUSD	ALGOPAX	ALGORUB	ALGOTRY	ALGOTUSD	ALGOUSDC
ALGOUSDT	ALICEBIDR	ALICEBNB	ALICEBTC	ALICEBUSD	ALICETRY
ALICEUSDT	ALPACABNB	ALPACABTC	ALPACABUSD	ALPACAUSDT	ALPHABNB
ALPHABTC	ALPHABUSD	ALPHAUSDT	ALPINEBTC	ALPINEBUSD	ALPINEEUR
ALPINETRY	ALPINEUSDT	ALTBNB	ALTBTC	ALTFDUSD	ALTTRY
ALTUSDC	ALTUSDT	AMBBNB	AMBBTC	AMBBUSD	AMBETH
AMBUSDT	AMPBNB	AMPBTC	AMPBUSD	AMPTRY	AMPUSDT
ANCBNB	ANCBTC	ANCBUSD	ANCUSDT	ANIMEBNB	ANIMEFDUSD
ANIMETRY	ANIMEUSDC	ANIMEUSDT	ANKRBNB	ANKRBTC	ANKRBUSD
ANKRPAX	ANKRTRY	ANKRTUSD	ANKRUSDC	ANKRUSDT	ANTBNB
ANTBTC	ANTBUSD	ANTUSDT	ANYBTC	ANYBUSD	ANYUSDT
APEAUD	APEBNB	APEBRL	APEBTC	APEBUSD	APEETH
APEEUR	APEFDUSD	APEGBP	APETRY	APEUSDC	APEUSDT
API3BNB	API3BTC	API3BUSD	API3TRY	API3USDC	API3USDT
APPCBNB	APPCBTC	APPCETH	APTBRL	APTBTC	APTBUSD
APTETH	APTEUR	APTFDUSD	APTJPY	APTTRY	APTUSDC
APTUSDT	ARBBTC	ARBETH	ARBEUR	ARBFDUSD	ARBNB
ARBRUB	ARBTC	ARBTRY	ARBTUSD	ARBUSD	ARBUSDC
ARBUSDT	ARDRBNB	ARDRBTC	ARDRETH	ARDRUSDT	ARFDUSD
ARKBTC	ARKBUSD	ARKETH	ARKMBNB	ARKMBTC	ARKMFDUSD
ARKMRUB	ARKMTRY	ARKMTUSD	ARKMUSDC	ARKMUSDT	ARKTRY
ARKUSDT	ARNBTC	ARNETH	ARPABNB	ARPABTC	ARPABUSD
ARPAETH	ARPARUB	ARPATRY	ARPAUSDT	ARTRY	ARUSDC
ARUSDT	ASRBTC	ASRBUSD	ASRTRY	ASRUSDT	ASTBTC
ASTERTRY	ASTERUSDC	ASTERUSDT	ASTETH	ASTRBTC	ASTRBUSD
ASTRETH	ASTRUSDT	ASTUSDT	ATABNB	ATABTC	ATABUSD
ATAUSDT	ATMBTC	ATMBUSD	ATMTRY	ATMUSDT	ATOMBIDR
ATOMBNB	ATOMBRL	ATOMBTC	ATOMBUSD	ATOMETH	ATOMEUR

ATOMFDUSD	ATOMPAX	ATOMTRY	ATOMTUSD	ATOMUSDC	ATOMUSDT
ATRY	AUCTIONBTC	AUCTIONBUSD	AUCTIONFDUSD	AUCTIONTRY	AUCTIONUSDC
AUCTIONUSDT	AUDBUSD	AUDIOBTC	AUDIOBUSD	AUDIOTRY	AUDIOUSDT
AUDUSDC	AUDUSDT	AUSDC	AUSDT	AUTOBTC	AUTOBUSD
AUTOUSDT	AVABNB	AVABTC	AVABUSD	AVAUSDT	AVAXAUD
AVAXBIDR	AVAXBNB	AVAXBRL	AVAXBTC	AVAXBUSD	AVAXETH
AVAXEUR	AVAXFDUSD	AVAXGBP	AVAXTRY	AVAXTUSD	AVAXUSDC
AVAXUSDT	AVNTTRY	AVNTUSDC	AVNTUSDT	AWEBTC	AWEUSDT
AXLBTC	AXLFDUSD	AXLTRY	AXLUSDT	AXSAUD	AXSBIDR
AXSBNB	AXSBRL	AXSBTC	AXSBUSD	AXSETH	AXSTRY
AXSUSDC	AXSUSDT	BABYBNB	BABYEUR	BABYFDUSD	BABYTRY
BABYUSDC	BABYUSDT	BADGERBTC	BADGERBUSD	BADGERUSDT	BAKEBNB
BAKEBTC	BAKEBUSD	BAKETRY	BAKEUSDT	BALBNB	BALBTC
BALBUSD	BALUSDT	BANANABNB	BANANABTC	BANANAFDUSD	BANANAS31USDC
BANANAS31USDT	BANANATRY	BANANAUSDC	BANANAUSDT	BANDBNB	BANDBTC
BANDBUSD	BANDTRY	BANDUSDT	BARBTC	BARBUSD	BARDBNB
BARDFDUSD	BARDTRY	BARDUSDC	BARDUSDT	BARTRY	BARUSDT
BATBNB	BATBTC	BATBUSD	BATETH	BATPAX	BATTUSD
BATUSDC	BATUSDT	BBBNB	BBBTC	BBFDUSD	BBTRY
BBUSDC	BBUSDT	BCCBNB	BCCBTC	BCCETH	BCCUSDT
BCDBTC	BCDETH	BCHABCBTC	BCHABCBUSD	BCHABCPAX	BCHABCTUSD
BCHABCUSDC	BCHABCUSDT	BCHABUSD	BCHBNB	BCHBTC	BCHBUSD
BCHEUR	BCHFDUSD	BCHJPY	BCHPAX	BCHSVBTC	BCHSVPAX
BCHSVTUSD	BCHSVUSDC	BCHSVUSDT	BCHTRY	BCHTUSD	BCHUSDC
BCHUSDT	BCNBNB	BCNBTC	BCNETH	BCPTBNB	BCPTBTC
BCPTETH	BCPTPAX	BCPTTUSD	BCPTUSDC	BDOTDOT	BEAMBNB
BEAMBTC	BEAMUSDT	BEAMXTRY	BEAMXUSDC	BEAMXUSDT	BEARBUSD
BEARUSDT	BELBNB	BELBTC	BELBUSD	BELETH	BELTRY

BELUSDT	BERABNB	BERABTC	BERAFDUSD	BERATRY	BERAUSDC
BERAUSDT	BETABNB	BETABTC	BETABUSD	BETAETH	BETAUSDT
BETHBUSD	BETHETH	BETHUSDT	BFUSDUSDC	BFUSDUSDT	BGBPUSDC
BICOBTC	BICOBUSD	BICOUSDT	BIFIBNB	BIFIBUSD	BIFIUSDT
BIGTIMEUSDC	BIGTIMEUSDT	BIOBNB	BIOFDUSD	BIOTRY	BIOUSDC
BIOUSDT	BKRWBUSD	BKRWUSDT	BLURBTC	BLURFDUSD	BLURTRY
BLURUSDC	BLURUSDT	BLZBNB	BLZBTC	BLZBUSD	BLZETH
BLZFDUSD	BLZUSDT	BMTBNB	BMTFDUSD	BMTTRY	BMTUSDC
BMTUSDT	BNBARS	BNBAUD	BNBBEARBUSD	BNBBEARUSDT	BNBBIDR
BNBBKRW	BNBBRL	BNBBTC	BNBBULLBUSD	BNBBULLUSDT	BNBBUSD
BNBDAI	BNBETH	BNBEUR	BNBFDUSD	BNBGBP	BNBIDRT
BNBJPY	BNBNGN	BNBPAX	BNBRUB	BNBTRY	BNBTUSD
BNBUAH	BNBUSDC	BNBUSDP	BNBUSDS	BNBUSDT	BNBUST
BNBZAR	BNSOLSOL	BNSOLUSDT	BNTBTC	BNTBUSD	BNTETH
BNTUSDT	BNXBNB	BNXBTC	BNXBUSD	BNXTRY	BNXUSDC
BNXUSDT	BOMEBTC	BOMEEUR	BOMEFDUSD	BOMETRY	BOMEUSDC
BOMEUSDT	BONDBNB	BONDBTC	BONDBUSD	BONDETH	BONDUSDT
BONKBRL	BONKFDUSD	BONKTRY	BONKUSDC	BONKUSDT	BOTBTC
BOTBUSD	BQXBTC	BQXETH	BRDBNB	BRDBTC	BRDETH
BROCCOLI714USDC	BROCCOLI714USDT	BSWBNB	BSWBUSD	BSWETH	BSWTRY
BSWUSDT	BTCAEUR	BTCARS	BTCAUD	BTCBBTC	BTCBIDR
BTCBKRW	BTCBRL	BTCBUSD	BTCDAI	BTCEUR	BTCEURI
BTCFDUSD	BTCGBP	BTCGYEN	BTCIDRT	BTCJPY	BTCMXN
BTCNGN	BTCPAX	BTCPLN	BTCRON	BTCRUB	BTCSTBTC
BTCSTBUSD	BTCSTUSDT	BTCTRY	BTCTUSD	BTCUAH	BTCUSDC
BTCUSDP	BTCUSDS	BTCUSDT	BTCUST	BTCVAI	BTCZAR
BTGBTC	BTGBUSD	BTGETH	BTGUSDT	BTSBNB	BTSBTC
BTSBUSD	BTSETH	BTSUSDT	BTTBNB	BTTBRL	BTTBTC

BTTBUSD	BTTCBUSD	BTTCTRY	BTTCUSDC	BTTCUSDT	BTTEUR
BTTPAX	BTTTRX	BTTTRY	BTTTUSD	BTTUSDC	BTTUSDT
BULLBUSD	BULLUSDT	BURGERBNB	BURGERBUSD	BURGERETH	BURGERUSDT
BUSDBIDR	BUSDBKRW	BUSDBRL	BUSDBVND	BUSDDAI	BUSDIDRT
BUSDNGN	BUSDPLN	BUSDRON	BUSDRUB	BUSDTRY	BUSDUAH
BUSDUSDT	BUSDVAI	BUSDZAR	BZRXBNB	BZRXBTC	BZRXBUSD
BZRXUSDT	C98BNB	C98BRL	C98BTC	C98BUSD	C98USDT
CAKEAUD	CAKEBNB	CAKEBRL	CAKEBTC	CAKEBUSD	CAKEGBP
CAKETRY	CAKETUSD	CAKEUSDC	CAKEUSDT	CATIBNB	CATIBRL
CATIFDUSD	CATITRY	CATIUSDC	CATIUSDT	CBNB	CDTBTC
CDTETH	CELOBTC	CELOBUSD	CELOTRY	CELOUSDT	CELRBNB
CELRBTC	CELRBUSD	CELRETH	CELRUSDT	CETUSTRY	CETUSUSDC
CETUSUSDT	CFDUSD	CFXBTC	CFXBUSD	CFXTRY	CFXTUSD
CFXUSDC	CFXUSDT	CGPTUSDC	CGPTUSDT	CHATBTC	CHATETH
CHESSBNB	CHESSBTC	CHESSBUSD	CHESSUSDC	CHESSUSDT	CHRBNB
CHRBTC	CHRBUSD	CHRETH	CHRUSDT	CHZBNB	CHZBRL
CHZBTC	CHZBUSD	CHZEUR	CHZFDUSD	CHZGBP	CHZTRY
CHZUSDC	CHZUSDT	CITYBNB	CITYBTC	CITYBUSD	CITYTRY
CITYUSDT	CKBBTC	CKBBUSD	CKBTRY	CKBUSDC	CKBUSDT
CLOAKBTC	CLOAKETH	CLVBNB	CLVBTC	CLVBUSD	CLVUSDT
CMTBNB	CMTBTC	CMTETH	CNDBNB	CNDBTC	CNDETH
COCOSBNB	COCOSBTC	COCOSBUSD	COCOSTRY	COCOSUSDT	COMBOBNB
COMBOTRY	COMBOUSDT	COMPBNB	COMPBTC	COMPBUSD	COMPTRY
COMPTUSD	COMPUSDC	COMPUSDT	COOKIEUSDC	COOKIEUSDT	COSBNB
COSBTC	COSBUSD	COSTRY	COSUSDT	COTIBNB	COTIBTC
COTIBUSD	COTITRY	COTIUSDT	COVERBUSD	COVERETH	COWTRY
COWUSDC	COWUSDT	CREAMBNB	CREAMBUSD	CREAMUSDT	CRVBNB
CRVBTC	CRVBUSD	CRVETH	CRVTRY	CRVUSDC	CRVUSDT

CTKBNB	CTKBTC	CTKBUSD	CTKUSDT	CTRY	CTSIBNB
CTSIBTC	CTSIBUSD	CTSIUSDT	CTXCBNB	CTXCBTC	CTXCBUSD
CTXCUSDT	CUSDC	CUSDT	CVCBNB	CVCBTC	CVCBUSD
CVCETH	CVCUSDC	CVCUSDT	CVPBUSD	CVPETH	CVPUSDT
CVXBTC	CVXBUSD	CVXUSDC	CVXUSDT	CYBERBNB	CYBERBTC
CYBERETH	CYBERFDUSD	CYBERTRY	CYBERTUSD	CYBERUSDC	CYBERUSDT
DAIBNB	DAIBTC	DAIBUSD	DAIJPY	DAIUSDT	DARBNB
DARBTC	DARBUSD	DARETH	DAREUR	DARTRY	DARUSDT
DASHBNB	DASHBTC	DASHBUSD	DASHETH	DASHUSDT	DATABTC
DATABUSD	DATAETH	DATAUSDT	DCRBNB	DCRBTC	DCRBUSD
DCRUSDT	DEGOBTC	DEGOBUSD	DEGOUSDT	DENTBTC	DENTBUSD
DENTETH	DENTTRY	DENTUSDT	DEXEBUSD	DEXEETH	DEXEUSDT
DFBUSD	DFETH	DFUSDC	DFUSDT	DGBBTC	DGBBUSD
DGBUSDT	DGDBTC	DGDETH	DIABNB	DIABTC	DIABUSD
DIAUSDT	DLTBNB	DLTBTC	DLTETH	DNTBTC	DNTBUSD
DNTETH	DNTUSDT	DOCKBTC	DOCKBUSD	DOCKETH	DOCKUSDT
DODOBTC	DODOBUSD	DODOUSDT	DOGEAUD	DOGEBIDR	DOGEBNB
DOGEBRL	DOGEBTC	DOGEBUSD	DOGEEUR	DOGEFDUSD	DOGEGBP
DOGEJPY	DOGEPAX	DOGERUB	DOGETRY	DOGETUSD	DOGEUSDC
DOGEUSDT	DOGSBNB	DOGSBRL	DOGSFDUSD	DOGSTRY	DOGSUSDC
DOGSUSDT	DOLOBNB	DOLOFDUSD	DOLOTRY	DOLOUSDC	DOLOUSDT
DOTAUD	DOTBIDR	DOTBKRW	DOTBNB	DOTBRL	DOTBTC
DOTBUSD	DOTETH	DOTEUR	DOTFDUSD	DOTGBP	DOTNGN
DOTRUB	DOTTRY	DOTTUSD	DOTUSDC	DOTUSDT	DREPBNB
DREPBTC	DREPBUSD	DREPUSDT	DTRY	DUSDT	DUSKBNB
DUSKBTC	DUSKBUSD	DUSKPAX	DUSKUSDC	DUSKUSDT	DYDXBNB
DYDXBTC	DYDXBUSD	DYDXETH	DYDXFDUSD	DYDXTRY	DYDXUSDC
DYDXUSDT	DYMBTC	DYMFDUSD	DYMTRY	DYMUSDT	EASYBTC

EASYETH	EDENBNB	EDENFDUSD	EDENTRY	EDENUSDC	EDENUSDT
EDOBTC	EDOETH	EDUBNB	EDUBTC	EDUEUR	EDUTRY
EDUTUSD	EDUUSDT	EGLDBNB	EGLDBTC	EGLDBUSD	EGLDETH
EGLDEUR	EGLDFDUSD	EGLDRON	EGLDTRY	EGLDUSDC	EGLDUSDT
EIGENBTC	EIGENFDUSD	EIGENTRY	EIGENUSDC	EIGENUSDT	ELFBTC
ELFBUSD	ELFETH	ELFUSDT	ENABNB	ENABRL	ENABTC
ENAEUR	ENAFDUSD	ENATRY	ENAUSDC	ENAUSDT	ENGBTC
ENGETH	ENJBNB	ENJBRL	ENJBTC	ENJBUSD	ENJETH
ENJEUR	ENJGBP	ENJTRY	ENJUSDC	ENJUSDT	ENSBNB
ENSBTC	ENSBUSD	ENSFDUSD	ENSOBNB	ENSOFDUSD	ENSOTRY
ENSOUSDC	ENSOUSDT	ENSTRY	ENSUSDC	ENSUSDT	EOSAUD
EOSBEARBUSD	EOSBEARUSDT	EOSBNB	EOSBTC	EOSBULLBUSD	EOSBULLUSDT
EOSBUSD	EOSETH	EOSEUR	EOSFDUSD	EOSPAX	EOSTRY
EOSTUSD	EOSUSDC	EOSUSDT	EPICUSDC	EPICUSDT	EPSBTC
EPSBUSD	EPSUSDT	EPXBUSD	EPXUSDT	ERABNB	ERAFDUSD
ERATRY	ERAUSDC	ERAUSDT	ERDBNB	ERDBTC	ERDBUSD
ERDPAX	ERDUSDC	ERDUSDT	ERNBNB	ERNBUSD	ERNUSDT
ETCBNB	ETCBRL	ETCBTC	ETCBUSD	ETCETH	ETCEUR
ETCFDUSD	ETCGBP	ETCPAX	ETCTRY	ETCTUSD	ETCUSDC
ETCUSDT	ETHAEUR	ETHARS	ETHAUD	ETHBEARBUSD	ETHBEARUSDT
ETHBIDR	ETHBKRW	ETHBRL	ETHBTC	ETHBULLBUSD	ETHBULLUSDT
ETHBUSD	ETHDAI	ETHEUR	ETHEURI	ETHFDUSD	ETHFIBNB
ETHFIBTC	ETHFIFDUSD	ETHFITRY	ETHFIUSDC	ETHFIUSDT	ETHGBP
ETHJPY	ETHMXN	ETHNGN	ETHPAX	ETHPLN	ETHRUB
ETHTRY	ETHTUSD	ETHUAH	ETHUSDC	ETHUSDP	ETHUSDT
ETHUST	ETHZAR	EULBNB	EULFDUSD	EULTRY	EULUSDC
EULUSDT	EURAEUR	EURBUSD	EUREURI	EURIUSDC	EURIUSDT
EURUSDC	EURUSDT	EVXBTC	EVXETH	EZBTC	EZETH

FARMBNB	FARMBTC	FARMBUSD	FARMETH	FARMUSDT	FDUSDBUSD
FDUSDTRY	FDUSDUSDC	FDUSDUSDT	FETBNB	FETBRL	FETBTC
FETBUSD	FETFDUSD	FETTRY	FETUSDC	FETUSDT	FFBNB
FFFDUSD	FFTRY	FFUSDC	FFUSDT	FIDABNB	FIDABTC
FIDABUSD	FIDATRY	FIDAUSDT	FILBNB	FILBTC	FILBUSD
FILETH	FILFDUSD	FILTRY	FILUSDC	FILUSDT	FIOBNB
FIOBTC	FIOBUSD	FIOUSDT	FIROBTC	FIROBUSD	FIROETH
FIROUSDT	FISBIDR	FISBRL	FISBTC	FISBUSD	FISTRY
FISUSDT	FLMBNB	FLMBTC	FLMBUSD	FLMUSDT	FLOKIFDUSD
FLOKITRY	FLOKITUSD	FLOKIUSDC	FLOKIUSDT	FLOWBNB	FLOWBTC
FLOWBUSD	FLOWUSDT	FLUXBTC	FLUXBUSD	FLUXUSDC	FLUXUSDT
FORBNB	FORBTC	FORBUSD	FORMTRY	FORMUSDC	FORMUSDT
FORTHBTC	FORTHBUSD	FORTHUSDT	FORUSDT	FRONTBTC	FRONTBUSD
FRONTETH	FRONTTRY	FRONTTUSD	FRONTUSDC	FRONTUSDT	FTMAUD
FTMBIDR	FTMBNB	FTMBRL	FTMBTC	FTMBUSD	FTMETH
FTMEUR	FTMFDUSD	FTMPAX	FTMRUB	FTMTRY	FTMTUSD
FTMUSDC	FTMUSDT	FTTBNB	FTTBTC	FTTBUSD	FTTETH
FTTUSDT	FUELBTC	FUELETH	FUNBNB	FUNBTC	FUNETH
FUNUSDC	FUNUSDT	FXSBTC	FXSBUSD	FXSUSDT	GALAAUD
GALABNB	GALABRL	GALABTC	GALABUSD	GALAETH	GALAEUR
GALAFDUSD	GALATRY	GALAUSDC	GALAUSDT	GALBNB	GALBRL
GALBTC	GALBUSD	GALETH	GALEUR	GALTRY	GALUSDT
GASBTC	GASBUSD	GASFDUSD	GASTRY	GASUSDT	GBPBUSD
GBPUSDT	GFTBUSD	GFTUSDT	GHSTBUSD	GHSTETH	GHSTUSDT
GLMBTC	GLMBUSD	GLMETH	GLMRBNB	GLMRBTC	GLMRBUSD
GLMRUSDT	GLMUSDT	GMTAUD	GMTBNB	GMTBRL	GMTBTC
GMTBUSD	GMTETH	GMTEUR	GMTGBP	GMTTRY	GMTUSDC
GMTUSDT	GMXBTC	GMXBUSD	GMXUSDC	GMXUSDT	GNOBNB

GNOBTC	GNOBUSD	GNOUSDT	GNSBTC	GNSUSDT	GNTBNB
GNTBTC	GNTETH	GOBNB	GOBTC	GPSBNB	GPSFDUSD
GPSTRY	GPSUSDC	GPSUSDT	GRSBTC	GRSETH	GRTBTC
GRTBUSD	GRTETH	GRTEUR	GRTFDUSD	GRTTRY	GRTUSDC
GRTUSDT	GTCBNB	GTCBTC	GTCBUSD	GTCUSDT	GTOBNB
GTOBTC	GTOBUSD	GTOETH	GTOPAX	GTOTUSD	GTOUSDC
GTOUSDT	GTRY	GUNBNB	GUNFDUSD	GUNTRY	GUNUSDC
GUNUSDT	GUSDT	GVTBTC	GVTETH	GXSBNB	GXSBTC
GXSETH	GXSUSDT	HAEDALBNB	HAEDALFDUSD	HAEDALTRY	HAEDALUSDC
HAEDALUSDT	HARDBNB	HARDBTC	HARDBUSD	HARDUSDT	HBARBNB
HBARBTC	HBARBUSD	HBARFDUSD	HBARTRY	HBARUSDC	HBARUSDT
HCBTC	HCETH	HCUSDT	HEGICBUSD	HEGICETH	HEIBTC
HEIUSDC	HEIUSDT	HEMIBNB	HEMIFDUSD	HEMITRY	HEMIUSDC
HEMIUSDT	HFTBTC	HFTBUSD	HFTUSDT	HIFIETH	HIFIUSDT
HIGHBNB	HIGHBTC	HIGHBUSD	HIGHTRY	HIGHUSDT	HIVEBNB
HIVEBTC	HIVEBUSD	HIVEFDUSD	HIVETRY	HIVEUSDC	HIVEUSDT
HMSTRBNB	HMSTRFDUSD	HMSTRTRY	HMSTRUSDC	HMSTRUSDT	HNTBTC
HNTBUSD	HNTUSDT	HOLOBNB	HOLOFDUSD	HOLOTRY	HOLOUSDC
HOLOUSDT	HOMEBNB	HOMEFDUSD	HOMETRY	HOMEUSDC	HOMEUSDT
HOOKBNB	HOOKBTC	HOOKBUSD	HOOKUSDT	HOTBNB	HOTBRL
HOTBTC	HOTBUSD	HOTETH	HOTEUR	HOTTRY	HOTUSDT
HSRBTC	HSRETH	HUMABNB	HUMAFDUSD	HUMATRY	HUMAUSDC
HUMAUSDT	HYPERBNB	HYPERFDUSD	HYPERTRY	HYPERUSDC	HYPERUSDT
ICNBTC	ICNETH	ICPBNB	ICPBTC	ICPBUSD	ICPETH
ICPEUR	ICPFDUSD	ICPRUB	ICPTRY	ICPUSDC	ICPUSDT
ICXBNB	ICXBTC	ICXBUSD	ICXETH	ICXUSDT	IDBNB
IDBTC	IDEUR	IDEXBNB	IDEXBTC	IDEXBUSD	IDEXFDUSD
IDEXUSDC	IDEXUSDT	IDTRY	IDTUSD	IDUSDT	ILVBNB

ILVBTC	ILVBUSD	ILVUSDC	ILVUSDT	IMXBNB	IMXBTC
IMXBUSD	IMXTRY	IMXUSDC	IMXUSDT	INITBNB	INITFDUSD
INITTRY	INITUSDC	INITUSDT	INJBNB	INJBTC	INJBUSD
INJETH	INJFDUSD	INJTRY	INJTUSD	INJUSDC	INJUSDT
INSBTC	INSETH	IOBNB	IOBTC	IOFDUSD	IOSTBTC
IOSTBUSD	IOSTETH	IOSTUSDT	IOTABNB	IOTABTC	IOTABUSD
IOTAETH	IOTAFDUSD	IOTATRY	IOTAUSDC	IOTAUSDT	IOTRY
IOTXBTC	IOTXBUSD	IOTXETH	IOTXJPY	IOTXUSDT	IOUSDC
IOUSDT	IQBNB	IQBUSD	IQUSDT	IRISBNB	IRISBTC
IRISBUSD	IRISUSDT	JASMYBNB	JASMYBTC	JASMYBUSD	JASMYETH
JASMYEUR	JASMYTRY	JASMYUSDT	JOEBTC	JOEBUSD	JOETRY
JOEUSDT	JSTBNB	JSTBTC	JSTBUSD	JSTUSDT	JTOFDUSD
JTOTRY	JTOUSDC	JTOUSDT	JUPFDUSD	JUPTRY	JUPUSDC
JUPUSDT	JUVBTC	JUVBUSD	JUVTRY	JUVUSDC	JUVUSDT
KAIAUSDC	KAIAUSDT	KAITOBNB	KAITOBRL	KAITOBTC	KAITOFDUSD
KAITOTRY	KAITOUSDC	KAITOUSDT	KAVABNB	KAVABTC	KAVABUSD
KAVAETH	KAVAUSDT	KDABTC	KDABUSD	KDAUSDT	KEEPBNB
KEEPBTC	KEEPBUSD	KEEPUSDT	KERNELBNB	KERNELFDUSD	KERNELTRY
KERNELUSDC	KERNELUSDT	KEYBTC	KEYBUSD	KEYETH	KEYUSDT
KLAYBNB	KLAYBTC	KLAYBUSD	KLAYUSDT	KMDBTC	KMDBUSD
KMDETH	KMDUSDT	KMNOUSDC	KMNOUSDT	KNCBNB	KNCBTC
KNCBUSD	KNCETH	KNCUSDT	KP3RBNB	KP3RBUSD	KP3RUSDT
KSMAUD	KSMBNB	KSMBTC	KSMBUSD	KSMETH	KSMTRY
KSMUSDT	LABNB	LAFDUSD	LATRY	LAUSDC	LAUSDT
LAYERBNB	LAYERBTC	LAYERFDUSD	LAYERTRY	LAYERUSDC	LAYERUSDT
LAZIOBTC	LAZIOBUSD	LAZIOEUR	LAZIOTRY	LAZIOUSDT	LDOBTC
LDOBUSD	LDOFDUSD	LDOTRY	LDOTUSD	LDOUSDC	LDOUSDT
LENDBKRW	LENDBTC	LENDBUSD	LENDETH	LENDUSDT	LEVERBUSD

LEVERTRY	LEVERUSDT	LINABNB	LINABTC	LINABUSD	LINAUSDT
LINEABNB	LINEAFDUSD	LINEATRY	LINEAUSDC	LINEAUSDT	LINKAUD
LINKBKRW	LINKBNB	LINKBRL	LINKBTC	LINKBUSD	LINKETH
LINKEUR	LINKFDUSD	LINKGBP	LINKJPY	LINKNGN	LINKPAX
LINKTRY	LINKTUSD	LINKUSDC	LINKUSDT	LISTABNB	LISTABRL
LISTAFDUSD	LISTATRY	LISTAUSDC	LISTAUSDT	LITBTC	LITBUSD
LITETH	LITUSDT	LOKABNB	LOKABTC	LOKABUSD	LOKAUSDT
LOOMBNB	LOOMBTC	LOOMBUSD	LOOMETH	LOOMTRY	LOOMUSDT
LPTBNB	LPTBTC	LPTBUSD	LPTJPY	LPTTRY	LPTUSDC
LPTUSDT	LQTYBTC	LQTYFDUSD	LQTYUSDT	LRCBNB	LRCBTC
LRCBUSD	LRCETH	LRCTRY	LRCUSDT	LSKBNB	LSKBTC
LSKBUSD	LSKETH	LSKUSDT	LTCBNB	LTCBRL	LTCBTC
LTCBUSD	LTCETH	LTCEUR	LTCFDUSD	LTCGBP	LTCJPY
LTCNGN	LTCPAX	LTCRUB	LTCTRY	LTCTUSD	LTCUAH
LTCUSDC	LTCUSDT	LTOBNB	LTOBTC	LTOBUSD	LTOUSDT
LUMIATRY	LUMIAUSDT	LUNAAUD	LUNABIDR	LUNABNB	LUNABRL
LUNABTC	LUNABUSD	LUNAETH	LUNAEUR	LUNAGBP	LUNATRY
LUNAUSDT	LUNAUST	LUNBTC	LUNCBUSD	LUNCTRY	LUNCUSDT
LUNETH	MAGICBTC	MAGICBUSD	MAGICFDUSD	MAGICTRY	MAGICUSDC
MAGICUSDT	MANABIDR	MANABNB	MANABRL	MANABTC	MANABUSD
MANAETH	MANATRY	MANAUSDT	MANTABNB	MANTABTC	MANTAFDUSD
MANTATRY	MANTAUSDC	MANTAUSDT	MASKBNB	MASKBUSD	MASKTRY
MASKUSDC	MASKUSDT	MATICAUD	MATICBIDR	MATICBNB	MATICBRL
MATICBTC	MATICBUSD	MATICETH	MATICEUR	MATICFDUSD	MATICGBP
MATICJPY	MATICRUB	MATICTRY	MATICTUSD	MATICUSDC	MATICUSDT
MAVBTC	MAVTRY	MAVTUSD	MAVUSDC	MAVUSDT	MBLBNB
MBLBTC	MBLBUSD	MBLUSDT	MBOXBNB	MBOXBTC	MBOXBUSD
MBOXTRY	MBOXUSDT	MCBNB	MCBTC	MCBUSD	MCOBNB

MCOBTC	MCOETH	MCOUSDT	MCUSDT	MDABTC	MDAETH
MDTBNB	MDTBTC	MDTBUSD	MDTUSDT	MDXBNB	MDXBTC
MDXBUSD	MDXUSDT	MEBTC	MEFDUSD	MEMEBNB	MEMEETH
MEMEFDUSD	MEMETRY	MEMETUSD	MEMEUSDC	MEMEUSDT	METISBTC
METISFDUSD	METISTRY	METISUSDT	METRY	MEUSDC	MEUSDT
MFTBNB	MFTBTC	MFTETH	MFTUSDT	MINABNB	MINABTC
MINABUSD	MINATRY	MINAUSDT	MIRABNB	MIRAFDUSD	MIRATRY
MIRAUSDC	MIRAUSDT	MIRBTC	MIRBUSD	MIRUSDT	MITHBNB
MITHBTC	MITHUSDT	MITOBNB	MITOFDUSD	MITOTRY	MITOUSDC
MITOUSDT	MKRBNB	MKRBTC	MKRBUSD	MKRTRY	MKRUSDC
MKRUSDT	MLNBNB	MLNBTC	MLNBUSD	MLNUSDT	MOBBTC
MOBBUSD	MOBUSDT	MODBTC	MODETH	MORPHOBNB	MORPHOFDUSD
MORPHOTRY	MORPHOUSDC	MORPHOUSDT	MOVEBNB	MOVEBTC	MOVEFDUSD
MOVETRY	MOVEUSDC	MOVEUSDT	MOVRBNB	MOVRBTC	MOVRBUSD
MOVRTRY	MOVRUSDT	MTHBTC	MTHETH	MTLBTC	MTLBUSD
MTLETH	MTLTRY	MTLUSDT	MUBARAKTRY	MUBARAKUSDC	MUBARAKUSDT
MULTIBTC	MULTIBUSD	MULTIUSDT	NANOBNB	NANOBTC	NANOBUSD
NANOETH	NANOUSDT	NASBNB	NASBTC	NASETH	NAVBNB
NAVBTC	NAVETH	NBSBTC	NBSUSDT	NCASHBNB	NCASHBTC
NCASHETH	NEARBNB	NEARBRL	NEARBTC	NEARBUSD	NEARETH
NEAREUR	NEARFDUSD	NEARJPY	NEARRUB	NEARTRY	NEARUSDC
NEARUSDT	NEBLBNB	NEBLBTC	NEBLBUSD	NEBLUSDT	NEIROBRL
NEIROEUR	NEIROFDUSD	NEIROJPY	NEIROTRY	NEIROUSDC	NEIROUSDT
NEOBNB	NEOBTC	NEOBUSD	NEOETH	NEOPAX	NEORUB
NEOTRY	NEOTUSD	NEOUSDC	NEOUSDT	NEWTBNB	NEWTFDUSD
NEWTTRY	NEWTUSDC	NEWTUSDT	NEXOBTC	NEXOBUSD	NEXOUSDT
NFPBNB	NFPBTC	NFPFDUSD	NFPTRY	NFPTUSD	NFPUSDT
NILBNB	NILFDUSD	NILTRY	NILUSDC	NILUSDT	NKNBNB

NKNBTC	NKNBUSD	NKNUSDT	NMRBTC	NMRBUSD	NMRTRY
NMRUSDC	NMRUSDT	NOMFDUSD	NOMTRY	NOMUSDC	NOMUSDT
NOTBNB	NOTBRL	NOTEUR	NOTFDUSD	NOTTRY	NOTUSDC
NOTUSDT	NPXSBTC	NPXSETH	NPXSUSDC	NPXSUSDT	NTRNBNB
NTRNBTC	NTRNTRY	NTRNUSDT	NUAUD	NUBNB	NUBTC
NUBUSD	NULSBNB	NULSBTC	NULSBUSD	NULSETH	NULSUSDT
NURUB	NUUSDT	NXPCBNB	NXPCFDUSD	NXPCTRY	NXPCUSDC
NXPCUSDT	NXSBNB	NXSBTC	NXSETH	OAXBTC	OAXETH
OAXUSDT	OCEANBNB	OCEANBTC	OCEANBUSD	OCEANTRY	OCEANUSDT
OGBTC	OGBUSD	OGNBNB	OGNBTC	OGNBUSD	OGNTRY
OGNUSDT	OGTRY	OGUSDT	OMBTC	OMBUSD	OMGBNB
OMGBTC	OMGBUSD	OMGETH	OMGUSDT	OMNIBNB	OMNIBRL
OMNIBTC	OMNIFDUSD	OMNITRY	OMNIUSDC	OMNIUSDT	OMTRY
OMUSDC	OMUSDT	ONDOTRY	ONDOUSDC	ONDOUSDT	ONEBIDR
ONEBNB	ONEBTC	ONEBUSD	ONEETH	ONEPAX	ONETRY
ONETUSD	ONEUSDC	ONEUSDT	ONGBNB	ONGBTC	ONGUSDT
ONTBNB	ONTBTC	ONTBUSD	ONTETH	ONTPAX	ONTTRY
ONTUSDC	ONTUSDT	OOKIBNB	OOKIBUSD	OOKIETH	OOKIUSDT
OPBNB	OPBTC	OPBUSD	OPENBNB	OPENFDUSD	OPENTRY
OPENUSDC	OPENUSDT	OPETH	OPEUR	OPFDUSD	OPTRY
OPTUSD	OPUSDC	OPUSDT	ORCAFDUSD	ORCATRY	ORCAUSDC
ORCAUSDT	ORDIBTC	ORDIFDUSD	ORDITRY	ORDITUSD	ORDIUSDC
ORDIUSDT	ORNBTC	ORNBUSD	ORNUSDT	OSMOBTC	OSMOBUSD
OSMOUSDC	OSMOUSDT	OSTBNB	OSTBTC	OSTETH	OXTBTC
OXTBUSD	OXTUSDT	PARTIBNB	PARTIFDUSD	PARTITRY	PARTIUSDC
PARTIUSDT	PAXBNB	PAXBTC	PAXBUSD	PAXETH	PAXGBNB
PAXGBTC	PAXGBUSD	PAXGTRY	PAXGUSDC	PAXGUSDT	PAXTUSD
PAXUSDT	PDABTC	PDAUSDT	PENDLEBTC	PENDLEFDUSD	PENDLETRY

PENDLETUSD	PENDLEUSDC	PENDLEUSDT	PENGUBNB	PENGUFDUSD	PENGUTRY
PENGUUSDC	PENGUUSDT	PEOPLEBNB	PEOPLEBTC	PEOPLEBUSD	PEOPLEETH
PEOPLEFDUSD	PEOPLETRY	PEOPLEUSDC	PEOPLEUSDT	PEPEBRL	PEPEEUR
PEPEFDUSD	PEPEJPY	PEPETRY	PEPETUSD	PEPEUSDC	PEPEUSDT
PERLBNB	PERLBTC	PERLUSDC	PERLUSDT	PERPBTC	PERPBUSD
PERPUSDT	PHABTC	PHABUSD	PHATRY	PHAUSDC	PHAUSDT
PHBBNB	PHBBTC	PHBBUSD	PHBPAX	PHBTRY	PHBTUSD
PHBUSDC	PHBUSDT	PHXBNB	PHXBTC	PHXETH	PIVXBNB
PIVXBTC	PIVXUSDT	PIXELBNB	PIXELBTC	PIXELFDUSD	PIXELTRY
PIXELUSDC	PIXELUSDT	PLABNB	PLABTC	PLABUSD	PLAUSDT
PLUMEBNB	PLUMEFDUSD	PLUMETRY	PLUMEUSDC	PLUMEUSDT	PNTBTC
PNTUSDT	PNUTBRL	PNUTBTC	PNUTEUR	PNUTFDUSD	PNUTTRY
PNUTUSDC	PNUTUSDT	POABNB	POABTC	POAETH	POEBTC
POEETH	POLBNB	POLBRL	POLBTC	POLETH	POLEUR
POLFDUSD	POLJPY	POLSBNB	POLSBTC	POLSBUSD	POLSUSDT
POLTRY	POLUSDC	POLUSDT	POLYBNB	POLYBTC	POLYBUSD
POLYUSDT	POLYXBTC	POLYXBUSD	POLYXTRY	POLYXUSDT	PONDBTC
PONDBUSD	PONDUSDT	PORTALBNB	PORTALBTC	PORTALFDUSD	PORTALTRY
PORTALUSDT	PORTOBTC	PORTOBUSD	PORTOEUR	PORTOTRY	PORTOUSDT
POWRBNB	POWRBTC	POWRBUSD	POWRETH	POWRUSDT	PPTBTC
PPTETH	PROMBNB	PROMBTC	PROMBUSD	PROMUSDT	PROSBUSD
PROSETH	PROSUSDT	PROVEBNB	PROVEFDUSD	PROVETRY	PROVEUSDC
PROVEUSDT	PSGBTC	PSGBUSD	PSGTRY	PSGUSDT	PUMPTRY
PUMPUSDC	PUMPUSDT	PUNDIXBUSD	PUNDIXETH	PUNDIXUSDC	PUNDIXUSDT
PYRBTC	PYRBUSD	PYRUSDT	PYTHBTC	PYTHFDUSD	PYTHTRY
PYTHUSDC	PYTHUSDT	QIBNB	QIBTC	QIBUSD	QIUSDT
QKCBTC	QKCBUSD	QKCETH	QKCUSDT	QLCBNB	QLCBTC
QLCETH	QNTBNB	QNTBTC	QNTBUSD	QNTUSDC	QNTUSDT

QSPBNB	QSPBTC	QSPETH	QTUMBNB	QTUMBTC	QTUMBUSD
QTUMETH	QTUMTRY	QTUMUSDC	QTUMUSDT	QUICKBNB	QUICKBTC
QUICKBUSD	QUICKTUSD	QUICKUSDT	RADBNB	RADBTC	RADBUSD
RADTRY	RADUSDT	RAMPBTC	RAMPBUSD	RAMPUSDT	RAREBNB
RAREBRL	RAREBTC	RAREBUSD	RARETRY	RAREUSDC	RAREUSDT
RAYBNB	RAYBUSD	RAYFDUSD	RAYTRY	RAYUSDC	RAYUSDT
RCNBNB	RCNBTC	RCNETH	RDNBNB	RDNBTC	RDNETH
RDNTBTC	RDNTTRY	RDNTTUSD	RDNTUSDT	REDBTC	REDFDUSD
REDTRY	REDUSDC	REDUSDT	REEFBIDR	REEFBTC	REEFBUSD
REEFTRY	REEFUSDT	REIBNB	REIBUSD	REIETH	REIUSDT
RENBNB	RENBTC	RENBTCBTC	RENBTCETH	RENBUSD	RENDERBRL
RENDERBTC	RENDEREUR	RENDERFDUSD	RENDERTRY	RENDERUSDC	RENDERUSDT
RENUSDT	REPBNB	REPBTC	REPBUSD	REPUSDT	REQBTC
REQBUSD	REQETH	REQUSDT	RESOLVBNB	RESOLVFDUSD	RESOLVTRY
RESOLVUSDC	RESOLVUSDT	REZBNB	REZBTC	REZFDUSD	REZTRY
REZUSDC	REZUSDT	RGTBNB	RGTBTC	RGTBUSD	RGTUSDT
RIFBTC	RIFUSDT	RLCBNB	RLCBTC	RLCBUSD	RLCETH
RLCUSDT	RNDRBRL	RNDRBTC	RNDRBUSD	RNDREUR	RNDRFDUSD
RNDRTRY	RNDRUSDC	RNDRUSDT	RONINBTC	RONINFDUSD	RONINTRY
RONINUSDT	ROSEBNB	ROSEBTC	ROSEBUSD	ROSEETH	ROSETRY
ROSEUSDC	ROSEUSDT	RPLBTC	RPLBUSD	RPLUSDC	RPLUSDT
RPXBNB	RPXBTC	RPXETH	RSRBNB	RSRBTC	RSRBUSD
RSRFDUSD	RSRTRY	RSRUSDC	RSRUSDT	RUNEAUD	RUNEBNB
RUNEBTC	RUNEBUSD	RUNEETH	RUNEEUR	RUNEFDUSD	RUNEGBP
RUNETRY	RUNETUSD	RUNEUSDC	RUNEUSDT	RVNBTC	RVNBUSD
RVNTRY	RVNUSDC	RVNUSDT	SAGABNB	SAGABTC	SAGAFDUSD
SAGATRY	SAGAUSDC	SAGAUSDT	SAHARABNB	SAHARAFDUSD	SAHARATRY
SAHARAUSDC	SAHARAUSDT	SALTBTC	SALTETH	SANDAUD	SANDBIDR

SANDBNB	SANDBRL	SANDBTC	SANDBUSD	SANDETH	SANDFDUSD
SANDTRY	SANDUSDC	SANDUSDT	SANTOSBRL	SANTOSBTC	SANTOSBUSD
SANTOSTRY	SANTOSUSDT	SBNB	SBTC	SCBTC	SCBUSD
SCETH	SCRBTC	SCRFDUSD	SCRTBTC	SCRTBUSD	SCRTETH
SCRTRY	SCRTUSDT	SCRUSDT	SCUSDT	SEIBNB	SEIBTC
SEIFDUSD	SEIJPY	SEITRY	SEITUSD	SEIUSDC	SEIUSDT
SETH	SEUR	SFDUSD	SFPBTC	SFPBUSD	SFPUSDT
SHELLBNB	SHELLBTC	SHELLFDUSD	SHELLTRY	SHELLUSDC	SHELLUSDT
SHIBAUD	SHIBBRL	SHIBBUSD	SHIBDOGE	SHIBEUR	SHIBFDUSD
SHIBGBP	SHIBJPY	SHIBRUB	SHIBTRY	SHIBTUSD	SHIBUAH
SHIBUSDC	SHIBUSDT	SIGNBNB	SIGNFDUSD	SIGNTRY	SIGNUSDC
SIGNUSDT	SKLBTC	SKLBUSD	SKLTRY	SKLUSDC	SKLUSDT
SKYBNB	SKYBTC	SKYCOINBNB	SKYCOINBTC	SKYCOINETH	SKYETH
SKYTRY	SKYUSDC	SKYUSDT	SLFBTC	SLFTRY	SLFUSDC
SLFUSDT	SLPBIDR	SLPBNB	SLPBUSD	SLPETH	SLPTRY
SLPUSDT	SNGLSBTC	SNGLSETH	SNMBTC	SNMBUSD	SNMETH
SNTBTC	SNTBUSD	SNTETH	SNTUSDT	SNXBNB	SNXBTC
SNXBUSD	SNXETH	SNXTRY	SNXUSDC	SNXUSDT	SOLARS
SOLAUD	SOLBIDR	SOLBNB	SOLBRL	SOLBTC	SOLBUSD
SOLETH	SOLEUR	SOLFDUSD	SOLGBP	SOLJPY	SOLMXN
SOLRUB	SOLTRY	SOLTUSD	SOLUSDC	SOLUSDT	SOLVBNB
SOLVFDUSD	SOLVTRY	SOLVUSDT	SOMIBNB	SOMIFDUSD	SOMITRY
SOMIUSDC	SOMIUSDT	SOPHBNB	SOPHFDUSD	SOPHTRY	SOPHUSDC
SOPHUSDT	SPARTABNB	SPELLBNB	SPELLBTC	SPELLBUSD	SPELLTRY
SPELLUSDT	SPKBNB	SPKFDUSD	SPKTRY	SPKUSDC	SPKUSDT
SRMBIDR	SRMBNB	SRMBTC	SRMBUSD	SRMUSDT	SSVBTC
SSVBUSD	SSVETH	SSVTUSD	SSVUSDC	SSVUSDT	STEEMBNB
STEEMBTC	STEEMBUSD	STEEMETH	STEEMUSDC	STEEMUSDT	STGBTC

STGBUSD	STGUSDT	STMXBTC	STMXBUSD	STMXETH	STMXTRY
STMXUSDT	STOBNB	STOFDUSD	STORJBTC	STORJBUSD	STORJETH
STORJTRY	STORJUSDT	STORMBNB	STORMBTC	STORMETH	STORMUSDT
STOTRY	STOUSDC	STOUSDT	STPTBNB	STPTBTC	STPTBUSD
STPTUSDT	STRATBNB	STRATBTC	STRATBUSD	STRATETH	STRATUSDT
STRAXBTC	STRAXBUSD	STRAXETH	STRAXTRY	STRAXUSDT	STRKBTC
STRKFDUSD	STRKTRY	STRKUSDC	STRKUSDT	STRY	STXBNB
STXBTC	STXBUSD	STXFDUSD	STXTRY	STXUSDC	STXUSDT
SUBBTC	SUBETH	SUIBNB	SUIBRL	SUIBTC	SUIEUR
SUIFDUSD	SUIJPY	SUITRY	SUITUSD	SUIUSDC	SUIUSDT
SUNBTC	SUNBUSD	SUNTRY	SUNUSDT	SUPERBTC	SUPERBUSD
SUPERFDUSD	SUPERTRY	SUPERUSDT	SUSDBTC	SUSDC	SUSDETH
SUSDT	SUSDUSDT	SUSHIBNB	SUSHIBTC	SUSHIBUSD	SUSHITRY
SUSHIUSDC	SUSHIUSDT	SWRVBNB	SWRVBUSD	SXPAUD	SXPBIDR
SXPBNB	SXPBTC	SXPBUSD	SXPEUR	SXPGBP	SXPTRY
SXPUSDT	SXTBNB	SXTFDUSD	SXTTRY	SXTUSDC	SXTUSDT
SYNBTC	SYNUSDC	SYNUSDT	SYRUPTRY	SYRUPUSDC	SYRUPUSDT
SYSBNB	SYSBTC	SYSBUSD	SYSETH	SYSUSDT	TAOBTC
TAOFDUSD	TAOTRY	TAOUSDC	TAOUSDT	TBUSD	TCTBNB
TCTBTC	TCTUSDT	TFUELBNB	TFUELBTC	TFUELBUSD	TFUELPAX
TFUELTUSD	TFUELUSDC	TFUELUSDT	THEBNB	THEBTC	THEFDUSD
THETABNB	THETABTC	THETABUSD	THETAETH	THETAEUR	THETAFDUSD
THETATRY	THETAUSDC	THETAUSDT	THETRY	THEUSDC	THEUSDT
TIABTC	TIAFDUSD	TIATRY	TIATUSD	TIAUSDC	TIAUSDT
TKOBIDR	TKOBTC	TKOBUSD	TKOUSDT	TLMBNB	TLMBTC
TLMBUSD	TLMFDUSD	TLMTRY	TLMUSDC	TLMUSDT	TNBBTC
TNBETH	TNSRBTC	TNSRFDUSD	TNSRTRY	TNSRUSDC	TNSRUSDT
TNTBTC	TNTETH	TOMOBNB	TOMOBTC	TOMOBUSD	TOMOTRY

TOMOUSDC	TOMOUSDT	TONBTC	TONFDUSD	TONTRY	TONUSDC
TONUSDT	TORNBNB	TORNBTC	TORNBUSD	TORNUSDT	TOWNSBNB
TOWNSFDUSD	TOWNSTRY	TOWNSUSDC	TOWNSUSDT	TRBBNB	TRBBTC
TRBBUSD	TRBFDUSD	TRBTRY	TRBUSDC	TRBUSDT	TREEBNB
TREEFDUSD	TREETRY	TREEUSDC	TREEUSDT	TRIBEBNB	TRIBEBTC
TRIBEBUSD	TRIBEUSDT	TRIGBNB	TRIGBTC	TRIGETH	TROYBNB
TROYBTC	TROYBUSD	TROYTRY	TROYUSDC	TROYUSDT	TRUBTC
TRUBUSD	TRUMPBRL	TRUMPEUR	TRUMPFDUSD	TRUMPJPY	TRUMPTRY
TRUMPUSDC	TRUMPUSDT	TRURUB	TRUTRY	TRUUSDT	TRXAUD
TRXBNB	TRXBTC	TRXBUSD	TRXETH	TRXEUR	TRXFDUSD
TRXJPY	TRXNGN	TRXPAX	TRXTRY	TRXTUSD	TRXUSDC
TRXUSDT	TRXXRP	TSTFDUSD	TSTTRY	TSTUSDC	TSTUSDT
TURBOFDUSD	TURBOTRY	TURBOUSDC	TURBOUSDT	TUSDBNB	TUSDBTC
TUSDBTUSD	TUSDBUSD	TUSDC	TUSDETH	TUSDT	TUSDTRY
TUSDUSDT	TUTUSDC	TUTUSDT	TVKBTC	TVKBUSD	TVKUSDT
TWTBTC	TWTBUSD	TWTTRY	TWTUSDC	TWTUSDT	UFTBUSD
UFTETH	UFTUSDT	UMABTC	UMABUSD	UMATRY	UMAUSDC
UMAUSDT	UNFIBNB	UNFIBTC	UNFIBUSD	UNFIETH	UNFITRY
UNFIUSDT	UNIAUD	UNIBNB	UNIBTC	UNIBUSD	UNIETH
UNIEUR	UNIFDUSD	UNITRY	UNIUSDC	UNIUSDT	USD1USDT
USDCBNB	USDCBRL	USDCBUSD	USDCCZK	USDCPAX	USDCPLN
USDCRON	USDCTRY	USDCTUSD	USDCUSDT	USDEUSDC	USDEUSDT
USDPBUSD	USDPUSDT	USDSBUSDS	USDSBUSDT	USDSPAX	USDSTUSD
USDSUSDC	USDSUSDT	USDTARS	USDTBIDR	USDTBKRW	USDTBRL
USDTBVND	USDTCOP	USDTCZK	USDTDAI	USDTGYEN	USDTIDRT
USDTMXN	USDTNGN	USDTPLN	USDTRON	USDTRUB	USDTTRY
USDTUAH	USDTVAI	USDTZAR	USTBTC	USTBUSD	USTCBUSD
USTCFDUSD	USTCTRY	USTCUSDT	USTUSDT	USUALBTC	USUALFDUSD

USUALTRY	USUALUSDC	USUALUSDT	UTKBTC	UTKBUSD	UTKUSDC
UTKUSDT	VANABNB	VANAFDUSD	VANATRY	VANAUSDC	VANAUSDT
VANRYBTC	VANRYTRY	VANRYUSDC	VANRYUSDT	VELODROMEUSDC	VELODROMEUSDT
VENBNB	VENBTC	VENETH	VENUSDT	VETBNB	VETBTC
VETBUSD	VETETH	VETEUR	VETGBP	VETTRY	VETUSDC
VETUSDT	VGXBTC	VGXETH	VGXUSDT	VIABNB	VIABTC
VIAETH	VIBBTC	VIBBUSD	VIBEBTC	VIBEETH	VIBETH
VIBUSDT	VICBTC	VICTRY	VICUSDT	VIDTBTC	VIDTBUSD
VIDTUSDT	VIRTUALTRY	VIRTUALUSDC	VIRTUALUSDT	VITEBNB	VITEBTC
VITEBUSD	VITEUSDT	VOXELBNB	VOXELBTC	VOXELBUSD	VOXELETH
VOXELUSDT	VTHOBNB	VTHOBUSD	VTHOTRY	VTHOUSDT	WABIBNB
WABIBTC	WABIETH	WALBNB	WALFDUSD	WALTRY	WALUSDC
WALUSDT	WANBNB	WANBTC	WANETH	WANUSDT	WAVESBNB
WAVESBTC	WAVESBUSD	WAVESETH	WAVESEUR	WAVESPAX	WAVESRUB
WAVESTRY	WAVESTUSD	WAVESUSDC	WAVESUSDT	WAXPBNB	WAXPBTC
WAXPBUSD	WAXPUSDT	WBETHETH	WBETHUSDT	WBTC	WBTCBTC
WBTCBUSD	WBTCETH	WBTCUSDT	WCTBNB	WCTFDUSD	WCTTRY
WCTUSDC	WCTUSDT	WFDUSD	WIFBRL	WIFBTC	WIFEUR
WIFFDUSD	WIFTRY	WIFUSDC	WIFUSDT	WINBNB	WINBRL
WINBTC	WINBUSD	WINEUR	WINGBNB	WINGBTC	WINGBUSD
WINGETH	WINGSBTC	WINGSETH	WINGUSDT	WINTRX	WINUSDC
WINUSDT	WLDBTC	WLDEUR	WLDFDUSD	WLDRUB	WLDTRY
WLDUSDC	WLDUSDT	WLFIBRL	WLFIEUR	WLFITRY	WLFIUSDC
WLFIUSDT	WNXMBNB	WNXMBTC	WNXMBUSD	WNXMUSDT	WOOBNB
WOOBTC	WOOBUSD	WOOUSDT	WPRBTC	WPRETH	WRXBNB
WRXBTC	WRXBUSD	WRXEUR	WRXUSDT	WTCBNB	WTCBTC
WTCETH	WTCUSDT	WTRY	WUSDC	WUSDT	XAIBNB
XAIBTC	XAIFDUSD	XAITRY	XAITUSD	XAIUSDC	XAIUSDT

XECBUSD	XECTRY	XECUSDT	XEMBNB	XEMBTC	XEMBUSD
XEMETH	XEMUSDT	XLMBNB	XLMBTC	XLMBUSD	XLMETH
XLMEUR	XLMFDUSD	XLMJPY	XLMPAX	XLMTRY	XLMTUSD
XLMUSDC	XLMUSDT	XMRBNB	XMRBTC	XMRBUSD	XMRETH
XMRUSDT	XNOBTC	XNOBUSD	XNOETH	XNOUSDT	XPLBNB
XPLFDUSD	XPLTRY	XPLUSDC	XPLUSDT	XRPAUD	XRPBEARBUSD
XRPBEARUSDT	XRPBIDR	XRPBKRW	XRPBNB	XRPBRL	XRPBTC
XRPBULLBUSD	XRPBULLUSDT	XRPBUSD	XRPETH	XRPEUR	XRPFDUSD
XRPGBP	XRPJPY	XRPMXN	XRPNGN	XRPPAX	XRPRUB
XRPTRY	XRPTUSD	XRPUSDC	XRPUSDT	XTZBNB	XTZBTC
XTZBUSD	XTZETH	XTZTRY	XTZUSDC	XTZUSDT	XUSDUSDT
XVGBTC	XVGBUSD	XVGETH	XVGTRY	XVGTUSD	XVGUSDT
XVSBNB	XVSBTC	XVSBUSD	XVSTRY	XVSUSDT	XZCBNB
XZCBTC	XZCETH	XZCUSDT	XZCXRP	YBBNB	YBFDUSD
YBTRY	YBUSDC	YBUSDT	YFIBNB	YFIBTC	YFIBUSD
YFIEUR	YFIIBNB	YFIIBTC	YFIIBUSD	YFIIUSDT	YFIUSDT
YGGBNB	YGGBTC	YGGBUSD	YGGTRY	YGGUSDC	YGGUSDT
YOYOBNB	YOYOBTC	YOYOETH	ZBTBNB	ZBTFDUSD	ZBTTRY
ZBTUSDC	ZBTUSDT	ZECBNB	ZECBTC	ZECBUSD	ZECETH
ZECPAX	ZECTUSD	ZECUSDC	ZECUSDT	ZENBNB	ZENBTC
ZENBUSD	ZENETH	ZENUSDC	ZENUSDT	ZILBIDR	ZILBNB
ZILBTC	ZILBUSD	ZILETH	ZILEUR	ZILTRY	ZILUSDT
ZKBTC	ZKCBNB	ZKCFDUSD	ZKCTRY	ZKCUSDC	ZKCUSDT
ZKFDUSD	ZKTRY	ZKUSDC	ZKUSDT	ZROBTC	ZROFDUSD
ZROTRY	ZROUSDC	ZROUSDT	ZRXBNB	ZRXBTC	ZRXBUSD
ZRXETH	ZRXUSDT				

Example Applications

The Binance Crypto Price dataset enables you to accurately design strategies for Cryptocurrencies. Examples include the following strategies:

- Buy and hold
- Trading Cryptocurrency volatility and price action
- Allocating a small portion of your portfolio to Cryptocurrencies to hedge against inflation

Classic Algorithm Example

The following example algorithm buys and holds Bitcoin through the Binance exchange:

```
from AlgorithmImports import * class CoinAPIDataAlgorithm(QCAlgorithm): def initialize(self) -> None: self.set_start_date(2024, 9, 1) self.set_end_date(2024, 12, 31) # Set the account currency to USDT, since you can't trade with # USD and the algorithm doesn't automatically convert USD & USDT # holdings. self.set_account_currency("USDT", 100000) self.universe_settings.asynchronous = True # Binance accepts both Cash and Margin account types. Select the # one you need for the best reality modeling. self.set_brokerage_model(BrokerageName.BINANCE, AccountType.MARGIN) # Add BTCUSDT data. self._btc = self.add_crypto("BTCUSDT", Resolution.MINUTE, Market.BINANCE) # Historical data history = self.history(self._btc.symbol, 30, Resolution.DAILY) # Add a Crypto universe that selects trading pairs on Binance. universe = self.add_universe(CryptoUniverse.binance(self._select_assets)) # Historical Universe data universe_history = self.history(universe, 30, Resolution.DAILY) for (universe_symbol, time), universe_day in universe_history.items(): for universe_item in universe_day: symbol = universe_item.symbol price = universe_item.close def _select_assets(self, universe_day): # Filter for materially traded Crypto pairs with significant # size and dollar volume, assuming higher capital flow in for # higher return. return [universe_item.symbol for universe_item in universe_day if universe_item.volume >= 100 and universe_item.volume_in_usd > 10_000_000] def on_data(self, slice: Slice) -> None: # Speculate-invest all available free cash on BTCUSDT, repecting # the order quantity restrictions to avoid invalid orders. if self.portfolio.cash_book['BTC'].amount != 0: return # Define a buffer so that we can scale down the order size, # avoiding trading errors. buffer = max( BinanceFeeModel.TAKER_TIER_1_FEE, self.settings.free_portfolio_value_percentage ) quantity = ( self.portfolio.cash_book['USDT'].amount / self._btc.ask_price * (1-buffer) ) # Check if the quantity exceeds the minimum order size. On # Binance, the minimum order size is measured in the quote # currency (USDT in this case). minimum_order_size = ( self._btc.symbol_properties.minimum_order_size / self._btc.ask_price ) if abs(quantity) < minimum_order_size: return self.market_order(self._btc, quantity)
```

```
public class CoinAPIDataAlgorithm : QCAlgorithm { private Crypto _btc; public override void Initialize() { SetStartDate(2024, 9, 1); SetEndDate(2024, 12, 31); // Set the account currency to USDT, since you can't trade with // USD and the algorithm doesn't automatically convert USD & USDT // holdings. SetAccountCurrency("USDT", 100000); UniverseSettings.Asynchronous = true; // Binance accepts both Cash and Margin account types. Select the // one you need for the best reality modeling. SetBrokerageModel(BrokerageName.Binance, AccountType.Margin); // Add BTCUSDT data. _btc = AddCrypto("BTCUSDT", Resolution.Minute, Market.Binance); // Historical data var history = History(_btc.Symbol, 30, Resolution.Daily); // Add a Crypto universe that selects trading pairs on Binance. var universe = AddUniverse(CryptoUniverse.Binance(SelectAssets)); // Historical Universe data var universeHistory = History(universe, 30, Resolution.Daily); foreach (var universeDay in universeHistory) { foreach (CryptoUniverse universeItem in universeDay) { var symbol = universeItem.Symbol; var price = universeItem.Close; } } } private IEnumerable<Symbol> SelectAssets(IEnumerable<CryptoUniverse> universeDay) { // Filter for materially traded Crypto pairs with significant // size and dollar volume, assuming higher capital flow in for // higher return. return from universeItem in universeDay where universeItem.Volume >= 100m && universeItem.VolumeInUsd > 10000000m select universeItem.Symbol; } public override void OnData(Slice slice) { // Speculate-invest all available free cash on BTCUSDT, repecting // the order quantity restrictions to avoid invalid orders. if (Portfolio.CashBook["BTC"].Amount != 0) { return; } // Define a buffer so that we can scale down the order size, // avoiding trading errors. var buffer = Math.Max(BinanceFeeModel.TakerTier1Fee, Settings.FreePortfolioValuePercentage); var quantity = Portfolio.CashBook["USDT"].Amount / _btc.AskPrice * (1-buffer); // Check if the quantity exceeds the minimum order size. On // Binance, the minimum order size is measured in the quote // currency (USDT in this case). var minimumOrderSize = _btc.SymbolProperties.MinimumOrderSize / _btc.AskPrice; if (Math.Abs(quantity) < minimumOrderSize) { return; } MarketOrder(_btc, quantity); } }
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of Crypto pairs on the Binance exchange and then forms a equal-weighted portfolio of all the pairs in the universe:

C#

```
public class CoinAPIDataAlgorithm : QCAlgorithm
{
    private List<string> _marketPairs;
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        Settings.FreePortfolioValuePercentage = 0.05m;

        // Set Account Currency to Tether, since USD and USDT will not auto-convert and USD cannot be used to trade
        SetAccountCurrency("USDT", 100000);
        // Binance accepts both Cash and Margin account types, select the one you need for the best reality modeling.
        SetBrokerageModel(BrokerageName.Binance, AccountType.Margin);

        // Warm up the security with the last known price to avoid conversion error
        Settings.SeedInitialPrices = true;
        UniverseSettings.Asynchronous = true;
        UniverseSettings.Resolution = Resolution.Minute;
        // Add Crypto Universe Selection that select crypto pairs in Binance exchange
        AddUniverse(CryptoUniverse.Binance(UniverseSelectionFilter));
        // Get the pairs on Binance that have USDT as the quote currency.
        _marketPairs = SymbolPropertiesDatabase.GetSymbolPropertiesList(Market.Binance)
            .Where(x => x.Value.QuoteCurrency == AccountCurrency)
            .Select(x => x.Key.Symbol)
            .ToList();

        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromMinutes(20), 0.025,
null));
        // Equally invest to evenly dissipate the capital concentration risk of inidividual crypto pair
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }

    private IEnumerable<Symbol> UniverseSelectionFilter(IEnumerable<CryptoUniverse> universeDay)
    {
        // Select pairs where the quote currency is USDT.
        return universeDay.Where(c => _marketPairs.Contains(c.Symbol.Value))
            // Filter for materially traded crypto pairs with significant size and dollar volume, assuming higher
capital flow in for higher return
            .OrderBy(c => c.VolumeInUsd)
            .TakeLast(3)
            .Select(c => c.Symbol);
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach(var security in changes.AddedSecurities)
        {
            // Historical data
            var history = History(security.Symbol, 30, Resolution.Daily);
        }
    }
}
```

Research Example

The following example lists crypto-currency pairs with the greatest dollar volume in the Binance exchange:

```C#
var qb = new QuantBook();

// Add Cryptocurrency pair
var symbol = qb.AddCrypto("BTCUSDT", market:Market.Binance).Symbol;

// Historical data
var history = qb.History(symbol, 30, Resolution.Daily);
foreach (var bar in history)
{
    Console.WriteLine($"{bar.EndTime} {bar}");
}

// Add Crypto Universe Selection
IEnumerable<Symbol> UniverseSelectionFilter(IEnumerable<CryptoUniverse> universeDay)
{
    return universeDay
        .Where(x => x.VolumeInUsd != null && x.VolumeInUsd > 10000m)
        .OrderByDescending(x => x.VolumeInUsd)
        .Take(5)
        .Select(x => x.Symbol);
}
var universe = qb.AddUniverse(CryptoUniverse.Binance(UniverseSelectionFilter));

// Historical Universe data
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var universeDay in universeHistory)
{
    Console.WriteLine($"=== {universeDay.First().EndTime} ===");
    foreach (CryptoUniverse universeItem in universeDay.OrderByDescending(x => x.VolumeInUsd))
    {
        Console.WriteLine($"{universeItem.Symbol}: {universeItem.VolumeInUsd}");
    }
}
```

Data Point Attributes

The Binance Crypto Price dataset provides `TradeBar` , `QuoteBar` , `Tick` , and `CryptoUniverse` objects.

TradeBar Attributes

`TradeBar` objects have the following attributes:

QuoteBar Attributes

`QuoteBar` objects have the following attributes:

Tick Attributes

`Tick` objects have the following attributes:

CryptoUniverse Attributes

`CryptoUniverse` objects have the following attributes:

CoinAPI

Binance US Crypto Price Data

Introduction

The Binance US Crypto Price Data by CoinAPI is for Cryptocurrency price and volume data points. The data covers 604 Cryptocurrency pairs, starts in October 2019, and is delivered on any frequency from tick to daily. This dataset is created by monitoring the trading activity on Binance US.

For more information about the Binance US Crypto Price Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

CoinAPI was founded by Artur Pietrzyk in 2016 with the goal of providing real-time and historical cryptocurrency market data, collected from hundreds of exchanges. CoinAPI provides access to Cryptocurrencies for traders, market makers, and developers building third-party applications.

Getting Started

The following snippets demonstrates how to set the brokerage model, request data, and perform universe selection with the Binance US dataset:

```C#
// Binance US only accepts cash accounts
SetBrokerageModel(BrokerageName.BinanceUS, AccountType.Cash);

_symbol = AddCrypto("BTCUSD", Resolution.Minute, Market.BinanceUS).Symbol;

_universe = AddUniverse(CryptoUniverse.BinanceUS(UniverseSelectionFilter));
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	October 2019
Asset Coverage	604 Cryptocurrency Pairs
Data Density	Dense
Resolution	Tick, Second, Minute, Hourly, & Daily
Timezone	UTC
Market Hours	Always Open

Requesting Data

To add Binance US Crypto Price data to your algorithm, call the `AddCrypto` method. Save a reference to the Crypto `Symbol` so

you can access the data later in your algorithm.

```C#
public class CoinAPIDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2020, 6, 1);
        SetEndDate(2021, 6, 1);
        SetCash(100000);

        // BinanceUS accepts Cash account type only, AccountType.Margin will result in an exception.
        SetBrokerageModel(BrokerageName.BinanceUS, AccountType.Cash);

        _symbol = AddCrypto("BTCUSD", Resolution.Minute, Market.BinanceUS).Symbol;
    }
}
```

For more information about creating Crypto subscriptions, see Requesting Data .

Accessing Data

To get the current Binance US Crypto Price data, index the `Bars` , `QuoteBars` , or `Ticks` properties of the current `Slice` with the Crypto `Symbol` . `Slice` objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```C#
public override void OnData(Slice slice)
{
    if (slice.Bars.ContainsKey(_symbol))
    {
        var tradeBar = slice.Bars[_symbol];
        Log($"{_symbol} price at {slice.Time}: {tradeBar.Close}");
    }

    if (slice.QuoteBars.ContainsKey(_symbol))
    {
        var quoteBar = slice.QuoteBars[_symbol];
        Log($"{_symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    if (slice.Ticks.ContainsKey(_symbol))
    {
        var ticks = slice.Ticks[_symbol];
        foreach (var tick in ticks)
        {
            Log($"{_symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

You can also iterate through all of the data objects in the current `Slice` .

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Bars)
    {
        var symbol = kvp.Key;
        var tradeBar = kvp.Value;
        Log($"{symbol} price at {slice.Time}: {tradeBar.Close}");
    }

    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        Log($"{symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    foreach (var kvp in slice.Ticks)
    {
        var symbol = kvp.Key;
        var ticks = kvp.Value;
        foreach (var tick in ticks)
        {
            Log($"{symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

For more information about accessing Crypto data, see Handling Data .

Historical Data

To get historical Binance US Crypto Price data, call the `History` method with the Crypto `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
// TradeBar objects
var historyTradeBars = History(_symbol, 100, Resolution.Daily);

// QuoteBar objects
var historyQuoteBars = History<QuoteBar>(_symbol, 100, Resolution.Minute);

// Tick objects
var historyTicks = History<Tick>(_symbol, TimeSpan.FromSeconds(10), Resolution.Tick);
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of Binance US Crypto pairs, call the `AddUniverse` method with a `CryptoUniverse` object. A Crypto universe uses a selection function to select Crypto pairs based on their OHLCV and dollar volume of the previous day as of midnight Coordinated Universal Time (UTC).

```
C#
public override void Initialize()
{
    UniverseSettings.Asynchronous = true;
    SetBrokerageModel(BrokerageName.BinanceUS, AccountType.Cash);
    _universe = AddUniverse(CryptoUniverse.BinanceUS(UniverseSelectionFilter));
}

private IEnumerable<Symol> UniverseSelectionFilter(IEnumerable<CryptoUniverse> universeDay)
{
    return from c in universeDay
           where c.Volume >= 100m && c.VolumeInUsd > 10000m
           select c.Symbol;
}
```

For more information about universe settings, see Settings .

Universe History

You can get historical universe data in an algorithm and in the Research Environment.

Historical Universe Data in Algorithms

To get historical universe data in an algorithm, call the `History` method with the `Universe` object, and the lookback period. If there is no data in the period you request, the history result is empty.

```C#
var history = History(_universe, 30, Resolution.Daily);
foreach (var universeDay in history)
{
    foreach (CryptoUniverse universeItem in universeDay)
    {
        Log($"{universeItem.Symbol} price at {universeItem.EndTime}: {universeItem.Close}");
    }
}
```

Historical Universe Data in Research

To get historical universe data in research, call the `UniverseHistory` method with the `Universe` object, and the lookback period. The `UniverseHistory` returns the filtered universe. If there is no data in the period you request, the history result is empty.

```C#
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var universeDay in universeHistory)
{
    foreach (CryptoUniverse universeItem in universeDay)
    {
        Console.WriteLine($"{universeItem.Symbol} price at {universeItem.EndTime}: {universeItem.Close}");
    }
}
```

You can call the `History` method in Research.

Remove Subscriptions

To unsubscribe from a Crypto pair that you added with the `AddCrypto` method, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_symbol);
```

The `RemoveSecurity` method cancels your open orders for the security and liquidates your holdings in the virtual pair .

Supported Assets

The following table shows the available Cryptocurrency pairs:

Pairs Available (604)					
1000MOGUSDT	1000REKTUSDT	1INCHUSD	1INCHUSD4	1INCHUSDT	A2ZUSDT
AAVEUSD	AAVEUSD4	AAVEUSDT	ACHUSD	ACHUSD4	ACHUSDT
ADABTC	ADABUSD	ADAETH	ADAUSD	ADAUSD4	ADAUSDC

ADAUSDT	ADXUSDT	AIXBTUSDT	ALGOBUSD	ALGOUSD	ALGOUSD4
ALGOUSDT	ALICEUSD	ALICEUSD4	ALICEUSDT	ALPINEUSD	ALPINEUSD4
ALPINEUSDT	AMPUSD	ANIMEUSDT	ANKRUSD	ANKRUSD4	ANKRUSDT
ANTUSD	ANTUSD4	ANTUSDT	APEUSD	APEUSD4	APEUSDT
API3USD	API3USD4	API3USDT	APTUSD	APTUSD4	APTUSDT
ARBUSD	ARBUSDT	ASTERUSDT	ASTRUSD	ASTRUSD4	ASTRUSDT
ATOMBTC	ATOMUSD	ATOMUSD4	ATOMUSDT	AUDIOUSD	AUDIOUSD4
AUDIOUSDT	AUSDT	AVAXBTC	AVAXUSD	AVAXUSD4	AVAXUSDT
AXLUSD	AXLUSD4	AXLUSDT	AXSUSD	AXSUSD4	AXSUSDT
BALUSD	BALUSD4	BALUSDT	BANDUSD	BANDUSD4	BANDUSDT
BATUSD	BATUSD4	BATUSDT	BCHBTC	BCHUSD	BCHUSD4
BCHUSDT	BICOUSD	BICOUSD4	BICOUSDT	BLURUSD	BLURUSDT
BNBBTC	BNBBUSD	BNBUSD	BNBUSD4	BNBUSDT	BNTUSD
BNTUSD4	BNTUSDT	BONDUSD	BONDUSD4	BONDUSDT	BONKUSD
BONKUSDT	BOSONUSD	BOSONUSD4	BOSONUSDT	BRETTUSDT	BTCBUSD
BTCDAI	BTCUSD	BTCUSD4	BTCUSDC	BTCUSDT	BTCUST
BTRSTUSD	BTRSTUSD4	BTRSTUSDT	BUSDUSD	BUSDUSD4	BUSDUSDT
CELOUSD	CELOUSD4	CELOUSDT	CELRUSD	CELRUSD4	CELRUSDT
CHZUSD	CHZUSD4	CHZUSDT	CLVUSD	CLVUSD4	CLVUSDT
COMPUSD	COMPUSD4	COMPUSDT	COTIUSD	COTIUSD4	COTIUSDT
CRVUSD	CRVUSD4	CRVUSDT	CTSIUSD	CTSIUSD4	CTSIUSDT
CUDOSUSDT	DAIUSD	DAIUSD4	DAIUSDT	DARUSD	DARUSD4
DARUSDT	DASHUSD	DASHUSD4	DASHUSDT	DATAUSDT	DGBUSD
DGBUSD4	DGBUSDT	DIAUSD	DIAUSD4	DIAUSDT	DOGEBTC
DOGEUSD	DOGEUSD4	DOGEUSDT	DOODUSDT	DOTBTC	DOTUSD
DOTUSD4	DOTUSDT	DUSDT	EGLDUSD	EGLDUSD4	EGLDUSDT
EIGENUSD	EIGENUSDT	ENAUSDT	ENJUSD	ENJUSD4	ENJUSDT
ENSUSD	ENSUSD4	ENSUSDT	EOSBUSD	EOSUSD	EOSUSD4

EOSUSDT	ETCUSD	ETCUSD4	ETCUSDT	ETHBTC	ETHBUSD
ETHDAI	ETHUSD	ETHUSD4	ETHUSDC	ETHUSDT	FARTCOINUSDT
FETUSD	FETUSD4	FETUSDT	FILUSD	FILUSD4	FILUSDT
FLOKI8USD	FLOKI8USDT	FLOKIUSD	FLOKIUSDT	FLOWUSD	FLOWUSD4
FLOWUSDT	FLUXUSD	FLUXUSD4	FLUXUSDT	FORTHUSD	FORTHUSD4
FORTHUSDT	FORTUSDT	FTMUSD	FTMUSD4	FTMUSDT	GALAUSD
GALAUSD4	GALAUSDT	GALUSD	GALUSD4	GALUSDT	GIGAUSDT
GLMUSD	GLMUSD4	GLMUSDT	GRTUSD	GRTUSD4	GRTUSDT
GTCUSD	GTCUSD4	GTCUSDT	GUSDT	HBARBUSD	HBARUSD
HBARUSD4	HBARUSDT	HNTUSD	HNTUSD4	HNTUSDT	HYPEUSD
HYPEUSDT	ICPUSD	ICPUSD4	ICPUSDT	ICXUSD	ICXUSD4
ICXUSDT	ILVUSD	ILVUSD4	ILVUSDT	IMXUSD	IMXUSD4
IMXUSDT	IOSTUSD	IOSTUSDT	IOTAUSD	IOTAUSD4	IOTAUSDT
IOTXUSDT	JAMUSD	JAMUSD4	JAMUSDT	JASMYUSD	JASMYUSD4
JASMYUSDT	JTOUSDT	JUPUSD	JUPUSDT	KAITOUSDT	KAVAUSD
KAVAUSD4	KAVAUSDT	KDAUSD	KDAUSD4	KDAUSDT	KNCUSD
KNCUSD4	KNCUSDT	KSHIBUSD	KSHIBUSD4	KSMUSD	KSMUSD4
KSMUSDT	LAYERUSDT	LAZIOUSD	LAZIOUSD4	LAZIOUSDT	LDOUSD
LDOUSD4	LDOUSDT	LINKBTC	LINKUSD	LINKUSD4	LINKUSDT
LOKAUSD	LOKAUSD4	LOKAUSDT	LOOMUSD	LOOMUSD4	LOOMUSDT
LPTBUSD	LPTUSD	LPTUSD4	LPTUSDT	LRCBTC	LRCUSD
LRCUSD4	LRCUSDT	LSKUSD	LSKUSD4	LSKUSDT	LTCBTC
LTCUSD	LTCUSD4	LTCUSDT	LTOUSD	LTOUSD4	LTOUSDT
MAGICUSDT	MANABTC	MANABUSD	MANAUSD	MANAUSD4	MANAUSDT
MASKUSD	MASKUSD4	MASKUSDT	MATICBTC	MATICBUSD	MATICETH
MATICUSD	MATICUSD4	MATICUSDT	METISUSD	METISUSDT	MEUSD
MEUSDT	MEWUSDT	MKRUSD	MKRUSD4	MKRUSDT	MOODENGUSDT
MXCUSD	MXCUSD4	MXCUSDT	NANOUSD	NEARBUSD	NEARUSD

NEARUSD4	NEARUSDT	NEIROUSDT	NEOUSD	NEOUSD4	NEOUSDT
NMRUSD	NMRUSD4	NMRUSDT	NOBODYUSDT	OCEANUSD	OCEANUSD4
OCEANUSDT	OGNUSD	OGNUSD4	OGNUSDT	OMGBUSD	OMGUSD
OMGUSD4	OMGUSDT	ONDOUSDT	ONEBUSD	ONEUSD	ONEUSD4
ONEUSDT	ONGUSDT	ONTUSD	ONTUSD4	ONTUSDT	OPUSD
OPUSD4	OPUSDT	ORBSUSDT	ORCAUSDT	OXTUSD	OXTUSD4
OXTUSDT	PAXGUSD	PAXGUSD4	PAXGUSDT	PENGUUSDT	PEPEUSD
PEPEUSDT	PNUTUSDT	POLBTC	POLETH	POLUSD	POLUSDT
POLYBTC	POLYBUSD	POLYUSD	POLYUSDT	POLYXUSD	POLYXUSD4
POLYXUSDT	PONDUSD	PONDUSD4	PONDUSDT	POPCATUSDT	PORTOUSD
PORTOUSD4	PORTOUSDT	PROMUSD	PROMUSD4	PROMUSDT	PUMPUSDT
QNTUSD	QNTUSD4	QNTUSDT	QTUMUSD	QTUMUSD4	QTUMUSDT
RADUSD	RADUSD4	RADUSDT	RAREUSD	RAREUSD4	RAREUSDT
REEFUSD	REEFUSD4	REEFUSDT	RENDERUSD	RENDERUSDT	RENUSD
RENUSD4	RENUSDT	REPBUSD	REPUSD	REQUSD	REQUSD4
REQUSDT	RLCUSD	RLCUSD4	RLCUSDT	RNDRUSD	RNDRUSD4
RNDRUSDT	ROSEUSD	ROSEUSD4	ROSEUSDT	RVNUSD	RVNUSD4
RVNUSDT	SANDUSD	SANDUSD4	SANDUSDT	SANTOSUSD	SANTOSUSD4
SANTOSUSDT	SHIBBUSD	SHIBUSD	SHIBUSDT	SKLUSD	SKLUSD4
SKLUSDT	SKYUSDT	SLPUSD	SLPUSD4	SLPUSDT	SNXUSD
SNXUSD4	SNXUSDT	SOLBTC	SOLBUSD	SOLETH	SOLUSD
SOLUSD4	SOLUSDC	SOLUSDT	SPELLUSD	SPELLUSD4	SPELLUSDT
SPXUSDT	SRMUSD	SRMUSDT	STGUSD	STGUSD4	STGUSDT
STMXUSD	STMXUSDT	STORJUSD	STORJUSD4	STORJUSDT	SUIUSD
SUIUSDT	SUSD	SUSDT	SUSHIUSD	SUSHIUSD4	SUSHIUSDT
SYSUSD	SYSUSD4	SYSUSDT	TFUELUSD	TFUELUSD4	TFUELUSDT
THETAUSD	THETAUSD4	THETAUSDT	TLMUSD	TLMUSD4	TLMUSDT
TOSHIUSDT	TRACUSD	TRACUSDT	TRUMPUSD	TRUMPUSDT	TRXBTC

TRXBUSD	TRXUSD	TRXUSD4	TRXUSDT	TURBOUSDT	TUSD
TUSD4	TUSDT	TUSDUSD	TUSDUSD4	TUSDUSDT	UNIBTC
UNIUSD	UNIUSD4	UNIUSDT	USDCBUSD	USDCUSD	USDCUSD4
USDCUSDT	USDTUSD	USDTUSD4	USELESSUSDT	USTUSD	USTUSDT
VETBTC	VETUSD	VETUSD4	VETUSDT	VIRTUALUSDT	VITEUSD
VITEUSD4	VITEUSDT	VOXELUSD	VOXELUSD4	VOXELUSDT	VTHOUSD
VTHOUSD4	VTHOUSDT	WAVESUSD	WAVESUSD4	WAVESUSDT	WAXPUSD
WAXPUSD4	WAXPUSDT	WBTCBTC	WIFUSDT	WLDUSDT	XDCUSDT
XECUSD	XECUSDT	XLMUSD	XLMUSD4	XLMUSDT	XNOUSD
XNOUSD4	XNOUSDT	XRPBTC	XRPBUSD	XRPUSD	XRPUSD4
XRPUSDT	XTZBTC	XTZBUSD	XTZUSD	XTZUSD4	XTZUSDT
YFIUSD	YFIUSD4	YFIUSDT	ZECUSD	ZECUSD4	ZECUSDT
ZENUSD	ZENUSD4	ZENUSDT	ZILBUSD	ZILUSD	ZILUSD4
ZILUSDT	ZRXUSD	ZRXUSD4	ZRXUSDT		

Example Applications

The Binance US Crypto Price dataset enables you to accurately design strategies for Cryptocurrencies. Examples include the following strategies:

- Buy and hold
- Trading Cryptocurrency volatility and price action
- Allocating a small portion of your portfolio to Cryptocurrencies to hedge against inflation

Classic Algorithm Example

The following example algorithm buys and holds Bitcoin through the Binance US exchange:

C#

```
public class CoinAPIDataAlgorithm : QCAlgorithm
{
    private Crypto _btc;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Set USDT as the account currency so we can trade BTCUSDT.
        SetAccountCurrency("USDT", 100000);
        UniverseSettings.Asynchronous = true;
        // Binance US accepts Cash account type only. AccountType.Margin
        // results in an exception.
        SetBrokerageModel(BrokerageName.BinanceUS, AccountType.Cash);
        // Add BTCUSDT data.
        _btc = AddCrypto("BTCUSDT", Resolution.Minute, Market.BinanceUS);
        // Historical data
        var history = History(_btc.Symbol, 30, Resolution.Daily);
        // Add a Crypto universe that selects trading pairs on Binance US.
        var universe = AddUniverse(CryptoUniverse.BinanceUS(SelectAssets));
        // Historical Universe data
        var universeHistory = History(universe, 30, Resolution.Daily);
        foreach (var universeDay in universeHistory)
        {
            foreach (CryptoUniverse universeItem in universeDay)
            {
                var symbol = universeItem.Symbol;
                var price = universeItem.Close;
            }
        }
    }

    private IEnumerable<Symbol> SelectAssets(IEnumerable<CryptoUniverse> universeDay)
    {
        // Filter for materially traded Crypto pairs with significant
        // size and dollar volume, assuming higher capital flow in for
        // higher return.
        return from universeItem in universeDay
                where universeItem.Volume >= 100m
                && universeItem.VolumeInUsd > 10000000m
                select universeItem.Symbol;
    }

    public override void OnData(Slice slice)
    {
        // Speculate-invest all available free cash on BTCUSDT, repecting
        // the order quantity restrictions to avoid invalid orders.
        if (Portfolio.CashBook["BTC"].Amount != 0)
        {
            return;
        }
        // Define a buffer so that we can scale down the order size,
        // avoiding trading errors.
        var buffer = Math.Max(BinanceFeeModel.TakerTier1Fee, Settings.FreePortfolioValuePercentage);
        var quantity = Portfolio.CashBook["USDT"].Amount / _btc.AskPrice * (1-buffer);
        // Check if the quantity exceeds the minimum order size. On
        // Binance US, the minimum order size is measured in the quote
        // currency (USDT in this case).
        var minimumOrderSize = _btc.SymbolProperties.MinimumOrderSize / _btc.AskPrice;
        if (Math.Abs(quantity) < minimumOrderSize)
        {
            return;
        }
        MarketOrder(_btc, quantity);
    }
}
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of Crypto pairs on the Binance US exchange and then forms a equal-weighted portfolio of all the pairs in the universe:

```C#
public class CoinAPIDataAlgorithm : QCAlgorithm
{
    private Symbol _btcusd;
    private List<string> _marketPairs;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);
        Settings.FreePortfolioValuePercentage = 0.05m;
        // Warm up the security with the last known price to avoid conversion error
        Settings.SeedInitialPrices = true;
        UniverseSettings.Asynchronous = true;
        UniverseSettings.Resolution = Resolution.Minute;
        // Add Crypto Universe Selection that select crypto pairs in BinanceUS exchange
        AddUniverse(CryptoUniverse.BinanceUS(UniverseSelectionFilter));
        // Get the pairs on Binance US that have USD as the quote currency.
        _marketPairs = SymbolPropertiesDatabase.GetSymbolPropertiesList(Market.BinanceUS)
            .Where(x => x.Value.QuoteCurrency == AccountCurrency)
            .Select(x => x.Key.Symbol)
            .ToList();
        AddAlpha( new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromMinutes(20), 0.025,
null) );
        // Equally invest to evenly dissipate the capital concentration risk of inidividual crypto pair
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }

    private IEnumerable<Symbol> UniverseSelectionFilter(IEnumerable<CryptoUniverse> universeDay)
    {
        // Select pairs where the quote currency is USD.
        return universeDay.Where(c => _marketPairs.Contains(c.Symbol.Value))
            // Filter for materially traded crypto pairs with significant size and dollar volume, assuming higher
capital flow in for higher return
            .OrderBy(c => c.VolumeInUsd)
            .TakeLast(3)
            .Select(c => c.Symbol);
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach(var security in changes.AddedSecurities)
        {
            // Historical data
            var history = History(security.Symbol, 30, Resolution.Daily);
            Debug($"We got {history.Count()} items from our history request");
        }
    }
}
```

Research Example

The following example lists crypto-currency pairs with the greatest dollar volume in the BinanceUS exchange:

```C#
var qb = new QuantBook();

// Add Cryptocurrency pair
var symbol = qb.AddCrypto("BTCUSDT", market:Market.BinanceUS).Symbol;

// Historical data
var history = qb.History(symbol, 30, Resolution.Daily);
foreach (var bar in history)
{
    Console.WriteLine($"{bar.EndTime} {bar}");
}

// Add Crypto Universe Selection
IEnumerable<Symbol> UniverseSelectionFilter(IEnumerable<CryptoUniverse> universeDay)
{
    return universeDay
        .Where(x => x.VolumeInUsd != null && x.VolumeInUsd > 10000m)
        .OrderByDescending(x => x.VolumeInUsd)
        .Take(5)
        .Select(x => x.Symbol);
}
var universe = qb.AddUniverse(CryptoUniverse.BinanceUS(UniverseSelectionFilter));

// Historical Universe data
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var universeDay in universeHistory)
{
    Console.WriteLine($"=== {universeDay.First().EndTime} ===");
    foreach (CryptoUniverse universeItem in universeDay.OrderByDescending(x => x.VolumeInUsd))
    {
        Console.WriteLine($"{universeItem.Symbol}: {universeItem.VolumeInUsd}");
    }
}
```

Data Point Attributes

The Binance US Crypto Price dataset provides `TradeBar` , `QuoteBar` , `Tick` , and `CryptoUniverse` objects.

TradeBar Attributes

`TradeBar` objects have the following attributes:

QuoteBar Attributes

`QuoteBar` objects have the following attributes:

Tick Attributes

`Tick` objects have the following attributes:

CryptoUniverse Attributes

`CryptoUniverse` objects have the following attributes:

CoinAPI

Bitfinex Crypto Price Data

Introduction

The Bitfinex Crypto Price Data by CoinAPI is for Cryptocurrency price and volume data points. The data covers 570 Cryptocurrency pairs, starts in January 2013, and is delivered on any frequency from tick to daily. This dataset is created by monitoring the trading activity on Bitfinex.

For more information about the Bitfinex Crypto Price Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

CoinAPI was founded by Artur Pietrzyk in 2016 with the goal of providing real-time and historical cryptocurrency market data, collected from hundreds of exchanges. CoinAPI provides access to Cryptocurrencies for traders, market makers, and developers building third-party applications.

Getting Started

The following snippet demonstrates how to request data from the Bitfinex Crypto Price dataset:

```C#
// Bitfinex accepts both Cash and Margin type account.
SetBrokerageModel(BrokerageName.Bitfinex, AccountType.Cash);
//SetBrokerageModel(BrokerageName.Bitfinex, AccountType.Margin);

_symbol = AddCrypto("BTCUSD", Resolution.Minute, Market.Bitfinex).Symbol;

_universe = AddUniverse(CryptoUniverse.Bitfinex(UniverseSelectionFilter));
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2013
Asset Coverage	570 Currency Pairs
Data Density	Dense
Resolution	Tick, Second, Minute, Hourly, & Daily
Timezone	UTC
Market Hours	Always Open

Requesting Data

To add Bitfinex Crypto Price data to your algorithm, call the `AddCrypto` method. Save a reference to the Crypto `Symbol` so you can access the data later in your algorithm.

```
C#
public class CoinAPIDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2020, 6, 1);
        SetEndDate(2021, 6, 1);
        SetCash(100000);

        // Bitfinex accepts both Cash and Margin type account.
        SetBrokerageModel(BrokerageName.Bitfinex, AccountType.Margin);

        _symbol = AddCrypto("BTCUSD", Resolution.Minute, Market.Bitfinex).Symbol;
    }
}
```

For more information about creating Crypto subscriptions, see Requesting Data .

Accessing Data

To get the current Bitfinex Crypto Price data, index the `Bars` , `QuoteBars` , or `Ticks` properties of the current `Slice` with the Crypto `Symbol` . `Slice` objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.Bars.ContainsKey(_symbol))
    {
        var tradeBar = slice.Bars[_symbol];
        Log($"{_symbol} price at {slice.Time}: {tradeBar.Close}");
    }

    if (slice.QuoteBars.ContainsKey(_symbol))
    {
        var quoteBar = slice.QuoteBars[_symbol];
        Log($"{_symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    if (slice.Ticks.ContainsKey(_symbol))
    {
        var ticks = slice.Ticks[_symbol];
        foreach (var tick in ticks)
        {
            Log($"{_symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

You can also iterate through all of the data objects in the current `Slice` .

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Bars)
    {
        var symbol = kvp.Key;
        var tradeBar = kvp.Value;
        Log($"{symbol} price at {slice.Time}: {tradeBar.Close}");
    }

    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        Log($"{symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    foreach (var kvp in slice.Ticks)
    {
        var symbol = kvp.Key;
        var ticks = kvp.Value;
        foreach (var tick in ticks)
        {
            Log($"{symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

For more information about accessing Crypto data, see Handling Data .

Historical Data

To get historical Bitfinex Crypto Price data, call the `History` method with the Crypto `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
// TradeBar objects
var historyTradeBars = History(_symbol, 100, Resolution.Daily);

// QuoteBar objects
var historyQuoteBars = History<QuoteBar>(_symbol, 100, Resolution.Minute);

// Tick objects
var historyTicks = History<Tick>(_symbol, TimeSpan.FromSeconds(10), Resolution.Tick);
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of Bitfinex Crypto pairs, call the `AddUniverse` method with a `CryptoUniverse` object. A Crypto universe uses a selection function to select Crypto pairs based on their OHLCV and dollar volume of the previous day as of midnight Coordinated Universal Time (UTC).

```
C#
public override void Initialize()
{
    UniverseSettings.Asynchronous = true;
    SetBrokerageModel(BrokerageName.Bitfinex, AccountType.Margin);
    _universe = AddUniverse(CryptoUniverse.Bitfinex(UniverseSelectionFilter));
}

private IEnumerable<Symbol> UniverseSelectionFilter(IEnumerable<CryptoUniverse> universeDay)
{
    return from c in universeDay
           where c.Volume >= 100m && c.VolumeInUsd > 10000m
           select c.Symbol;
}
```

For more information about universe settings, see Settings .

Universe History

You can get historical universe data in an algorithm and in the Research Environment.

Historical Universe Data in Algorithms

To get historical universe data in an algorithm, call the `History` method with the `Universe` object, and the lookback period. If there is no data in the period you request, the history result is empty.

```C#
var history = History(_universe, 30, Resolution.Daily);
foreach (var universeDay in history)
{
    foreach (CryptoUniverse universeItem in universeDay)
    {
        Log($"{universeItem.Symbol} price at {universeItem.EndTime}: {universeItem.Close}");
    }
}
```

Historical Universe Data in Research

To get historical universe data in research, call the `UniverseHistory` method with the `Universe` object, and the lookback period. The `UniverseHistory` returns the filtered universe. If there is no data in the period you request, the history result is empty.

```C#
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var universeDay in universeHistory)
{
    foreach (CryptoUniverse universeItem in universeDay)
    {
        Console.WriteLine($"{universeItem.Symbol} price at {universeItem.EndTime}: {universeItem.Close}");
    }
}
```

You can call the `History` method in Research.

Remove Subscriptions

To unsubscribe from a Crypto pair that you added with the `AddCrypto` method, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_symbol);
```

The `RemoveSecurity` method cancels your open orders for the security and liquidates your holdings in the virtual pair .

Data Points Attributes

The Bitfinex Crypto Price dataset provides `TradeBar` , `QuoteBar` , `Tick` , and `CryptoUniverse` objects.

TradeBar Attributes

`TradeBar` objects have the following attributes:

QuoteBar Attributes

`QuoteBar` objects have the following attributes:

Tick Attributes

`Tick` objects have the following attributes:

CryptoUniverse Attributes

`CryptoUniverse` objects have the following attributes:

Supported Assets

The following table shows the available Cryptocurrency pairs:

Pairs Available (570)					
1INCHUSD	1INCHUSDT	AAABBB	AAVEUSD	AAVEUSDT	ADABTC
ADAUSD	ADAUSDT	AIOZUSD	AIXUSD	AIXUSDT	ALBTUSD
ALBTUSDT	ALGOBTC	ALGOUSD	ALGOUSDT	ALT11M2507USD	ALT11M2507USDT
ALT11M250830USD	ALT11M250830USDT	ALT11M251029USD	ALT11M251029USDT	ALT2612USD	ALT2612USDT
AMPLBTC	AMPLUSD	AMPLUSDT	ANCUSD	ANCUSDT	ANTBTC
ANTETH	ANTUSD	APENFTUSD	APEUSD	APEUSDT	APTUSD
APTUSDT	ARBUSD	ARBUSDT	ATHUSD	ATHUSDT	ATOMBTC
ATOMETH	ATOMUSD	ATOMUSDT	AUSDTUSD	AUSDTUSDT	AVAXBTC
AVAXUSD	AVAXUSDT	AXSUSD	AXSUSDT	AZEROUSD	AZEROUSDT
B21USD	B21USDT	B2MUSD	B2MUSDT	BALUSD	BALUSDT
BANDUSD	BANDUSDT	BATUSD	BATUSDT	BCHABCUSD	BCHNUSD
BESTUSD	BFTUSD	BGBUSD	BGBUSDT	BLASTUSD	BLASTUSDT
BMIUSD	BMIUSDT	BNTUSD	BOBAUSD	BOBAUSDT	BONKUSD
BONKUSDT	BOSONUSD	BOSONUSDT	BSVBTC	BSVUSD	BTCCNHT
BTCEUR	BTCEURQ	BTCEURR	BTCEURT	BTCGBP	BTCJPY
BTCMIM	BTCTRY	BTCUSD	BTCUSDQ	BTCUSDR	BTCUSDT
BTCXAUT	BTCXCHF	BTGBTC	BTGUSD	BTSEUSD	BTTUSD
BXNUSD	BXNUSDT	CCDBTC	CCDUSD	CCDUSDT	CELOUSD
CELOUSDT	CELUSD	CELUSDT	CHEXUSD	CHSBBTC	CHSBUSD
CHSBUSDT	CHZUSD	CHZUSDT	CLOUSD	CNHCNHT	CNHTUSD
COMPUSD	COMPUSDT	CRVUSD	CRVUSDT	CTKUSD	CTKUSDT
DAIBTC	DAIETH	DAIUSD	DASHBTC	DASHUSD	DATAUSD

DCRUSD	DGBUSD	DOGEBTC	DOGEUSD	DOGEUSDT	DOPUSD
DOPUSDT	DORAUSD	DORAUSDT	DOTBTC	DOTUSD	DOTUSDT
DUSKBTC	DUSKUSD	DVFUSD	DYMUSD	DYMUSDT	EDOUSD
EGLDUSD	EGLDUSDT	EIGENUSD	EIGENUSDT	ENAUSD	ENAUSDT
ENJUSD	EOSBTC	EOSETH	EOSEUR	EOSGBP	EOSJPY
EOSUSD	EOSUSDT	ESSUSD	ETCBTC	ETCUSD	ETCUSDT
ETH2ETH	ETH2USD	ETH2USDT	ETHBTC	ETHEUR	ETHEURT
ETHGBP	ETHJPY	ETHUSD	ETHUSDT	ETHWUSD	ETHWUSDT
ETHXAUT	ETPBTC	ETPUSD	EURQEUR	EURQUSD	EURQUSDT
EURREUR	EURRUSD	EURRUSDT	EURSUSD	EURTEUR	EURTUSD
EURTUSDT	EURUSDT	EXOUSD	FCLUSD	FCLUSDT	FETUSD
FETUSDT	FILUSD	FILUSDT	FLOKIUSD	FLOKIUSDT	FORTHUSD
FORTHUSDT	FTMUSD	FTMUSDT	FTTUSD	FTTUSDT	FUNUSD
GALAUSD	GALAUSDT	GBPEURT	GBPUSDT	GLMUSD	GMMTUSD
GMMTUSDT	GNOUSD	GOMININGUSD	GOMININGUSDT	GOTEUR	GOTUSD
GRTUSD	GRTUSDT	GTXUSD	GTXUSDT	GUNUSD	GUNUSDT
HILSVUSD	HIXUSD	HIXUSDT	HMSTRUSD	HMSTRUSDT	HMTUSD
HMTUSDT	HTXDAOUSD	HTXDAOUSDT	HTXUSD	HTXUSDT	ICEUSD
ICPBTC	ICPUSD	ICPUSDT	IDUSD	IDUSDT	IOTABTC
IOTAETH	IOTAEUR	IOTAGBP	IOTAJPY	IOTAUSD	IQXUSD
IQXUSDT	JASMYUSD	JASMYUSDT	JPYUSDT	JSTBTC	JSTUSD
JSTUSDT	JUPUSD	JUPUSDT	JUSTICEUSD	JUSTICEUSDT	JXXUSD
JXXUSDT	KAIAUSD	KAIAUSDT	KAIUSD	KAIUSDT	KANUSD
KANUSDT	KARATEUSD	KARATEUSDT	KAVAUSD	KAVAUSDT	KMNOUSD
KMNOUSDT	KNCBTC	KNCUSD	KSMUSD	KSMUSDT	LDOUSD
LDOUSDT	LEOBTC	LEOEOS	LEOETH	LEOUSD	LEOUSDT
LIFIIIUSD	LIFIIIUSDT	LINKUSD	LINKUSDT	LRCUSD	LTCBTC
LTCUSD	LTCUSDT	LUNABTC	LUNAETH	LUNAUSD	LUNAUSDT

LYMUSD	MANABTC	MANAUSD	MANEKIUSD	MANEKIUSDT	MATICBTC
MATICUSD	MATICUSDT	MDOGBTC	MDOGUSD	MDOGUSDT	MEWUSD
MEWUSDT	MIMUSD	MIMUSDT	MIRUSD	MIRUSDT	MKRUSD
MKRUSDT	MLNUSD	MOBUSD	MOBUSDT	MPCUSD	MPCUSDT
MXNTUSD	NEARUSD	NEARUSDT	NEOBTC	NEOETH	NEOEUR
NEOGBP	NEOJPY	NEOUSD	NEOUSDT	NEXOBTC	NEXOUSD
NEXOUSDT	NOTUSD	NOTUSDT	NYMUSD	NYMUSDT	OCEANUSD
OCEANUSDT	ODEUSD	OMGBTC	OMGETH	OMGUSD	OMNIUSD
OPXUSD	OPXUSDT	ORSUSD	OXYUSD	OXYUSDT	PASSUSD
PAXUSD	PAXUSDT	PEPEUSD	PEPEUSDT	PLANETSUSD	PLANETSUSDT
PLUUSD	PNGUSD	PNGUSDT	PNKETH	PNKUSD	POLUSD
POLUSDT	QRDOUSD	QRDOUSDT	QSHUSD	QTFBTC	QTFUSD
QTUMBTC	QTUMUSD	RBTCBTC	RBTCUSD	REEFUSD	REEFUSDT
REP2BTC	REP2USD	REQUSD	ROSEUSD	ROSEUSDT	RRTUSD
SANBTC	SANDUSD	SANDUSDT	SANETH	SANUSD	SCROLLUSD
SCROLLUSDT	SEIUSD	SEIUSDT	SGBUSD	SGBUSDT	SHIBUSD
SHIBUSDT	SHMUSD	SHMUSDT	SNTUSD	SNXUSD	SNXUSDT
SOLBTC	SOLUSD	SOLUSDT	SONICUSD	SONICUSDT	SPECUSD
SPECUSDT	SPELLMIM	SPELLUSD	SPELLUSDT	SRMUSD	SRMUSDT
STGUSD	STGUSDT	STORJUSD	STRKUSD	STRKUSDT	STXUSD
STXUSDT	SUIUSD	SUIUSDT	SUKUUSD	SUKUUSDT	SUNUSD
SUNUSDT	SUSHIUSD	SUSHIUSDT	SWEATUSD	SWEATUSDT	SXXUSD
SXXUSDT	TERRAUSTUSD	TERRAUSTUSDT	TESTADATESTUSD	TESTALGOTESTUSD	TESTAPTTESTUSD
TESTAVAXTESTUSD	TESTBTCTESTUSD	TESTBTCTESTUSDT	TESTDOGETESTUSD	TESTDOTTESTUSD	TESTEOSTESTUSD
TESTETHTESTUSD	TESTFILTESTUSD	TESTLTCTESTUSD	TESTMATICTESTUSD	TESTMATICTESTUSDT	TESTNEARTESTUSD
TESTSOLTESTUSD	TESTXAUTTESTUSD	TESTXTZTESTUSD	THETAUSD	THETAUSDT	TIAUSD
TIAUSDT	TITAN1GBP	TITAN1USD	TITAN2GBP	TITAN2USD	TLOSUSD
TLOSUSDT	TOKENUSD	TOKENUSDT	TOMIUSD	TOMIUSDT	TONUSD

TONUSDT	TRADEUSD	TRADEUSDT	TRXBTC	TRXETH	TRXEUR
TRXUSD	TRXUSDT	TRYUSDT	TURBOUSD	TURBOUSDT	TUSDUSD
TUSDUSDT	UNIUSD	UNIUSDT	UOPUSD	UOPUSDT	UOSBTC
UOSUSD	USDCUSD	USDCUSDT	USDFUSD	USDFUSDT	USDQUSD
USDQUSDT	USDRUSD	USDRUSDT	USDTCNHT	USDTMXNT	USDTUSD
USTBLUSD	USTBLUSDT	UTKUSD	UXLINKUSD	UXLINKUSDT	VEEUSD
VELARUSD	VELARUSDT	VELOUSD	VELOUSDT	VETBTC	VETUSD
VETUSDT	VSYBTC	VSYUSD	WAVESUSD	WAVESUSDT	WAXUSD
WBTCBTC	WBTCUSD	WHBTUSD	WHBTUSDT	WIFUSD	WIFUSDT
WILDUSD	WILDUSDT	WNCGUSD	WNCGUSDT	WOOUSD	WOOUSDT
XAUTBTC	XAUTUSD	XAUTUSDT	XCHFUSD	XDCUSD	XDCUSDT
XLMBTC	XLMETH	XLMUSD	XLMUSDT	XMRBTC	XMRUSD
XMRUSDT	XRAUSD	XRDBTC	XRDUSD	XRPBTC	XRPUSD
XRPUSDT	XSNUSD	XTPUSD	XTPUSDT	XTZBTC	XTZUSD
XTZUSDT	XVGUSD	YFIUSD	YFIUSDT	ZCNUSD	ZECBTC
ZECUSD	ZILBTC	ZILUSD	ZKXUSD	ZKXUSDT	ZMTUSD
ZMTUSDT	ZROUSD	ZROUSDT	ZRXBTC	ZRXETH	ZRXUSD

Example Applications

The Bitfinex Crypto Price dataset enables you to accurately design strategies for Cryptocurrencies. Examples include the following strategies:

- Buy and hold
- Trading Cryptocurrency volatility and price action
- Allocating a small portion of your portfolio to Cryptocurrencies to hedge against inflation

Classic Algorithm Example

The following example algorithm buys and holds Bitcoin through the Bitfinex exchange:

C#

```
public class CoinAPIDataAlgorithm : QCAlgorithm
{
    private Crypto _btc;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        UniverseSettings.Asynchronous = true;
        SetCash(100000);
        // Bitfinex accepts both Cash and Margin account types. Select the
        // one you need for the best reality modeling.
        SetBrokerageModel(BrokerageName.Bitfinex, AccountType.Margin);
        // Add BTCUSD data.
        _btc = AddCrypto("BTCUSD", Resolution.Minute, Market.Bitfinex);
        // Historical data
        var history = History(_btc.Symbol, 30, Resolution.Daily);
        // Add a Crypto universe that selects trading pairs on Bitfinex.
        var universe = AddUniverse(CryptoUniverse.Bitfinex(SelectAssets));
        // Historical Universe data
        var universeHistory = History(universe, 30, Resolution.Daily);
        foreach (var universeDay in universeHistory)
        {
            foreach (CryptoUniverse universeItem in universeDay)
            {
                var symbol = universeItem.Symbol;
                var price = universeItem.Close;
            }
        }
    }

    private IEnumerable<Symbol> SelectAssets(IEnumerable<CryptoUniverse> universeDay)
    {
        // Filter for materially traded Crypto pairs with significant
        // size and dollar volume, assuming higher capital flow in for
        // higher return.
        return from universeItem in universeDay
                where universeItem.Volume >= 100m
                && universeItem.VolumeInUsd > 10000000m
                select universeItem.Symbol;
    }

    public override void OnData(Slice slice)
    {
        // Speculate-invest all available free cash on BTCUSD, repecting
        // the order quantity restrictions to avoid invalid orders.
        if (Portfolio.CashBook["BTC"].Amount != 0)
        {
            return;
        }
        // Define a buffer so that we can scale down the order size,
        // avoiding trading errors.
        var buffer = Math.Max(BitfinexFeeModel.TakerFee, Settings.FreePortfolioValuePercentage);
        var quantity = Portfolio.CashBook["USD"].Amount / _btc.AskPrice * (1-buffer);
        // Check if the quantity exceeds the minimum order size. On
        // Bitfinex, the minimum order size is measured in the base
        // currency (BTC in this case).
        if (Math.Abs(quantity) < _btc.SymbolProperties.MinimumOrderSize)
        {
            return;
        }
        MarketOrder(_btc, quantity);
    }
}
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of Crypto pairs on the Bitfinex exchange and then forms a equal-weighted portfolio of all the pairs in the universe:

```C#
public class CoinAPIDataAlgorithm : QCAlgorithm
{
    private List<string> _marketPairs;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        Settings.FreePortfolioValuePercentage = 0.05m;
        // Bitfinex accepts both Cash and Margin account types, select the one you need for the best reality
modeling.
        SetBrokerageModel(BrokerageName.Bitfinex, AccountType.Margin);

        // Warm up the security with the last known price to avoid conversion error
        Settings.SeedInitialPrices = true;
        UniverseSettings.Asynchronous = true;
        UniverseSettings.Resolution = Resolution.Minute;
        // Add Crypto Universe Selection that select crypto pairs in Bitfinex exchange
        AddUniverse(CryptoUniverse.Bitfinex(SelectAssets));
        // Get the pairs on Bitfinex that have USD as the quote currency.
        _marketPairs = SymbolPropertiesDatabase.GetSymbolPropertiesList(Market.Bitfinex)
            .Where(x => x.Value.QuoteCurrency == AccountCurrency)
            .Select(x => x.Key.Symbol)
            .ToList();
        AddAlpha( new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromHours(4), 0.025, null)
);
        // Equally invest to evenly dissipate the capital concentration risk of inidividual crypto pair
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }

    private IEnumerable<Symbol> SelectAssets(IEnumerable<CryptoUniverse> universeDay)
    {
        // Select pairs where the quote currency is USD.
        return universeDay.Where(c => _marketPairs.Contains(c.Symbol.Value))
            // Filter for materially traded crypto pairs with significant size and dollar volume, assuming higher
capital flow in for higher return
            .OrderBy(c => c.VolumeInUsd)
            .TakeLast(3)
            .Select(c => c.Symbol);
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach(var security in changes.AddedSecurities)
        {
            // Historical data
            var history = History(security.Symbol, 30, Resolution.Daily);
            Debug($"We got {history.Count()} items from our history request");
        }
    }
}
```

Research Example

The following example lists crypto-currency pairs with the greatest dollar volume in the Bitfinex exchange:

```C#
var qb = new QuantBook();

// Add Cryptocurrency pair
var symbol = qb.AddCrypto("BTCUSDT", market:Market.Bitfinex).Symbol;

// Historical data
var history = qb.History(symbol, 30, Resolution.Daily);
foreach (var bar in history)
{
    Console.WriteLine($"{bar.EndTime} {bar}");
}

// Add Crypto Universe Selection
IEnumerable<Symbol> UniverseSelectionFilter(IEnumerable<CryptoUniverse> universeDay)
{
    return universeDay
        .Where(x => x.VolumeInUsd != null && x.VolumeInUsd > 10000m)
        .OrderByDescending(x => x.VolumeInUsd)
        .Take(5)
        .Select(x => x.Symbol);
}
var universe = qb.AddUniverse(CryptoUniverse.Bitfinex(UniverseSelectionFilter));

// Historical Universe data
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var universeDay in universeHistory)
{
    Console.WriteLine($"=== {universeDay.First().EndTime} ===");
    foreach (CryptoUniverse universeItem in universeDay.OrderByDescending(x => x.VolumeInUsd))
    {
        Console.WriteLine($"{universeItem.Symbol}: {universeItem.VolumeInUsd}");
    }
}
```

Data Point Attributes

The Bitfinex Crypto Price dataset provides `TradeBar` , `QuoteBar` , `Tick` , and `CryptoUniverse` objects.

TradeBar Attributes

`TradeBar` objects have the following attributes:

QuoteBar Attributes

`QuoteBar` objects have the following attributes:

Tick Attributes

`Tick` objects have the following attributes:

CryptoUniverse Attributes

`CryptoUniverse` objects have the following attributes:

CoinAPI

Bybit Crypto Future Price Data

Introduction

The Bybit Crypto Future Price Data by CoinAPI is for Cryptocurrency Futures price and volume data points. The data covers 631 Cryptocurrency pairs, starts in August 2020, and is delivered on any frequency from tick to daily. This dataset is created by monitoring the trading activity on Bybit.

The Bybit Crypto Future Margin Rate Data dataset provides margin interest rate data to model margin costs.

For more information about the Bybit Crypto Future Price Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

CoinAPI was founded by Artur Pietrzyk in 2016 with the goal of providing real-time and historical cryptocurrency market data, collected from hundreds of exchanges. CoinAPI provides access to Cryptocurrencies for traders, market makers, and developers building third-party applications.

Getting Started

The following snippet demonstrates how to request data from the Bybit Crypto Future Price dataset:

```C#
private Symbol _cryptoFutureSymbol;

public override void Initialize
{
    SetBrokerageModel(BrokerageName.Bybit, AccountType.Margin);

    _cryptoFutureSymbol = AddCryptoFuture("BTCUSDT", Resolution.Minute).Symbol;
}
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	October 2019
Asset Coverage	631 Crypto Futures Pairs
Data Density	Dense
Resolution	Tick, Second, Minute, Hourly, & Daily
Timezone	UTC
Market Hours	Always Open

Requesting Data

To add Bybit Crypto Future Price data to your algorithm, call the `AddCryptoFuture` method. Save a reference to the Crypto Future `Symbol` so you can access the data later in your algorithm.

```
C#
public class CoinAPIDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2020, 6, 1);
        SetEndDate(2021, 6, 1);

        // Set Account Currency to Tether
        SetAccountCurrency("USDT", 100000);

        SetBrokerageModel(BrokerageName.Bybit, AccountType.Margin);

        var cryptoFuture = AddCryptoFuture("BTCUSDT", Resolution.Minute);
        _symbol = cryptoFuture.Symbol;
    }
}
```

For more information about creating Crypto Future subscriptions, see Requesting Data .

Accessing Data

To get the current Bybit Crypto Future Price data, index the `Bars` , `QuoteBars` , or `Ticks` properties of the current `Slice` with the Crypto Future `Symbol` . `Slice` objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.Bars.ContainsKey(_symbol))
    {
        var tradeBar = slice.Bars[_symbol];
        Log($"{_symbol} price at {slice.Time}: {tradeBar.Close}");
    }

    if (slice.QuoteBars.ContainsKey(_symbol))
    {
        var quoteBar = slice.QuoteBars[_symbol];
        Log($"{_symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    if (slice.Ticks.ContainsKey(_symbol))
    {
        var ticks = slice.Ticks[_symbol];
        foreach (var tick in ticks)
        {
            Log($"{_symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

You can also iterate through all of the data objects in the current `Slice` .

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Bars)
    {
        var symbol = kvp.Key;
        var tradeBar = kvp.Value;
        Log($"{symbol} price at {slice.Time}: {tradeBar.Close}");
    }

    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        Log($"{symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    foreach (var kvp in slice.Ticks)
    {
        var symbol = kvp.Key;
        var ticks = kvp.Value;
        foreach (var tick in ticks)
        {
            Log($"{symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

For more information about accessing Crypto Future data, see Handling Data .

Historical Data

To get historical Bybit Crypto Future Price data, call the `History` method with the Crypto Future `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
// TradeBar objects
var historyTradeBars = History(_symbol, 100, Resolution.Daily);

// QuoteBar objects
var historyQuoteBars = History<QuoteBar>(_symbol, 100, Resolution.Minute);

// Tick objects
var historyTicks = History<Tick>(_symbol, TimeSpan.FromSeconds(10), Resolution.Tick);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To unsubscribe from a Crypto Future contract that you added with the `AddCryptoFuture` method, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_symbol);
```

The `RemoveSecurity` method cancels your open orders for the security and liquidates your Crypto Future holdings.

Supported Assets

The following table shows the available Crypto Future pairs:

Pairs Available (631)

1CATUSDT	1INCHUSDT	A8USDT	AAVEUSD	AAVEUSDT	ACEUSDT
ACHUSDT	ACTUSDT	ACXUSDT	ADAUSD	ADAUSDT	AERGOUSDT
AEROUSDT	AEVOUSDT	AGIUSDT	AGIXUSDT	AGLDUSDT	AI16ZUSDT
AIDOGEUSDT	AIOZUSDT	AIUSDT	AIXBTUSDT	AKROUSDT	AKTUSDT
ALCHUSDT	ALEOUSDT	ALGOUSDT	ALICEUSDT	ALPACAUSDT	ALPHAUSDT
ALTUSDT	ALUUSDT	AMBUSDT	ANIMEUSDT	ANKRUSDT	ANTUSDT
APEUSDT	APEXUSDT	API3USDT	APTUSD	APTUSDT	APUUSDT
ARBUSDT	ARCUSDT	ARKMUSDT	ARKUSDT	ARPAUSDT	ARUSDT
ASTRUSDT	ATAUSDT	ATHUSDT	ATOMUSDT	AUCTIONUSDT	AUDIOUSDT
AVAAIUSDT	AVAILUSDT	AVAUSDT	AVAXUSD	AVAXUSDT	AVLUSDT
AXLUSDT	AXSUSDT	B3USDT	BABYDOGEUSDT	BABYUSDT	BADGERUSDT
BAKEUSDT	BALUSDT	BANANAS31USDT	BANANAUSDT	BANDUSDT	BANKUSDT
BANUSDT	BATUSDT	BBUSDT	BCHUSD	BCHUSDT	BEAMUSDT
BEERUSDT	BELUSDT	BENDOGUSDT	BERAUSDT	BICOUSDT	BIGTIMEUSDT
BILLYUSDT	BIOUSDT	BLASTUSDT	BLUEUSDT	BLURUSDT	BLZUSDT
BMTUSDT	BNBUSDT	BNTUSDT	BNXUSDT	BOBAUSDT	BOMEUSDT
BONDUSDT	BONKUSDT	BRETTUSDT	BROCCOLIUSDT	BRUSDT	BSVUSDT
BSWUSDT	BTCUSD	BTCUSDT	BTTUSDT	BUSDUSDT	BUZZUSDT
C98USDT	CAKEUSDT	CARVUSDT	CATIUSDT	CATSUSDT	CATUSDT
CEEKUSDT	CELOUSDT	CELRUSDT	CETUSUSDT	CFXUSDT	CGPTUSDT
CHEEMSUSDT	CHESSUSDT	CHILLGUYUSDT	CHRUSDT	CHZUSDT	CKBUSDT
CLANKERUSDT	CLOUDUSDT	COMBOUSDT	COMPUSDT	COOKIEUSDT	COOKUSDT
COQUSDT	COREUSDT	COSUSDT	COTIUSDT	COVALUSDT	COWUSDT
CPOOLUSDT	CROUSDT	CRVUSDT	CTCUSDT	CTKUSDT	CTSIUSDT
CVCUSDT	CVXUSDT	CYBERUSDT	DAOUSDT	DARKUSDT	DARUSDT
DASHUSDT	DATAUSDT	DBRUSDT	DEEPUSDT	DEGENUSDT	DENTUSDT
DEXEUSDT	DGBUSDT	DODOUSDT	DOGEUSD	DOGEUSDT	DOGSUSDT
DOGUSDT	DOODUSDT	DOP1USDT	DOTUSD	DOTUSDT	DRIFTUSDT

DUCKUSDT	DUSKUSDT	DYDXUSDT	DYMUSDT	EDUUSDT	EGLDUSDT
EIGENUSDT	ELONUSDT	ELXUSDT	ENAUSDT	ENJUSDT	ENSUSDT
EOSUSD	EOSUSDT	EPICUSDT	EPTUSDT	ETCUSD	ETCUSDT
ETHBTCUSDT	ETHFIUSDT	ETHUSD	ETHUSDT	ETHWUSDT	FARTCOINUSDT
FBUSDT	FDUSDUSDT	FETUSDT	FHEUSDT	FIDAUSDT	FILUSD
FILUSDT	FIOUSDT	FIREUSDT	FITFIUSDT	FLMUSDT	FLOCKUSDT
FLOKIUSDT	FLOWUSDT	FLRUSDT	FLUXUSDT	FORMUSDT	FORTHUSDT
FOXYUSDT	FRONTUSDT	FTMUSDT	FTNUSDT	FUELUSDT	FUNUSDT
FUSDT	FWOGUSDT	FXSUSDT	GALAUSDT	GALUSDT	GASUSDT
GEMSUSDT	GFTUSDT	GIGAUSDT	GLMRUSDT	GLMUSDT	GMEUSDT
GMTUSDT	GMXUSDT	GNOUSDT	GOATUSDT	GODSUSDT	GOMININGUSDT
GORKUSDT	GPSUSDT	GPTUSDT	GRASSUSDT	GRIFFAINUSDT	GRTUSDT
GTCUSDT	GUNUSDT	GUSDT	HAEDALUSDT	HBARUSDT	HEIUSDT
HFTUSDT	HIFIUSDT	HIGHUSDT	HIPPOUSDT	HIVEUSDT	HMSTRUSDT
HNTUSDT	HOOKUSDT	HOTUSDT	HPOS10IUSDT	HYPERUSDT	HYPEUSDT
ICPUSDT	ICXUSDT	IDEXUSDT	IDUSDT	ILVUSDT	IMXUSDT
INITUSDT	INJUSDT	IOSTUSDT	IOTAUSDT	IOTXUSDT	IOUSDT
IPUSDT	IQ50USDT	JASMYUSDT	JELLYJELLYUSDT	JOEUSDT	JSTUSDT
JTOUSDT	JUPUSDT	JUSDT	KAIAUSDT	KAITOUSDT	KASUSDT
KAVAUSDT	KDAUSDT	KERNELUSDT	KEYUSDT	KLAYUSDT	KMNOUSDT
KNCUSDT	KOMAUSDT	KSMUSDT	L3USDT	LADYSUSDT	LAIUSDT
LDOUSDT	LEVERUSDT	LINAUSDT	LINKUSD	LINKUSDT	LISTAUSDT
LITUSDT	LOOKSUSDT	LOOMUSDT	LPTUSDT	LQTYUSDT	LRCUSDT
LSKUSDT	LTCUSD	LTCUSDT	LTOUSDT	LUCEUSDT	LUMIAUSDT
LUNA2USDT	LUNCUSDT	MAGICUSDT	MAJORUSDT	MANAUSD	MANAUSDT
MANEKIUSDT	MANTAUSDT	MAPOUSDT	MASAUSDT	MASKUSDT	MATICUSDT
MAVIAUSDT	MAVUSDT	MAXUSDT	MBLUSDT	MBOXUSDT	MCUSDT
MDTUSDT	MELANIAUSDT	MEMEFIUSDT	MEMEUSDT	MERLUSDT	METISUSDT

MEUSDT	MEWUSDT	MICHIUSDT	MILKUSDT	MINAUSDT	MKRUSDT
MLNUSDT	MNTUSD	MNTUSDT	MOBILEUSDT	MOCAUSDT	MOGUSDT
MONUSDT	MOODENGUSDT	MORPHOUSDT	MOTHER1USDT	MOTHER2USDT	MOTHERUSDT
MOVEUSDT	MOVRUSDT	MTLUSDT	MUBARAKUSDT	MULTIUSDT	MUMUUSDT
MVLUSDT	MYRIAUSDT	MYROUSDT	NCUSDT	NEARUSD	NEARUSDT
NEIROCTOUSDT	NEIROETHUSDT	NEOUSDT	NFPUSDT	NFTUSDT	NILUSDT
NKNUSDT	NMRUSDT	NOTUSDT	NSUSDT	NTRNUSDT	NULSUSDT
NYANUSDT	OBOLUSDT	OBTUSDT	OCEANUSDT	OGNUSDT	OGUSDT
OKBUSDT	OLUSDT	OMGUSDT	OMNIUSDT	OMUSDT	ONDOUSDT
ONEUSDT	ONGUSDT	ONTUSDT	OPUSD	OPUSDT	ORBSUSDT
ORCAUSDT	ORDERUSDT	ORDIUSDT	ORNUSDT	OSMOUSDT	OXTUSDT
PARTIUSDT	PAXGUSDT	PEAQUSDT	PEIPEIUSDT	PENDLEUSDT	PENGUSDT
PENGUUSDT	PEOPLEUSDT	PEPEUSDT	PERPUSDT	PHAUSDT	PHBUSDT
PIPPINUSDT	PIRATEUSDT	PIXELUSDT	PIXFIUSDT	PLUMEUSDT	PNUTUSDT
POLUSDT	POLYXUSDT	PONKEUSDT	POPCATUSDT	PORTALUSDT	POWRUSDT
PRCLUSDT	PRE05EOSUSDT	PRE08EOSUSDT	PRIMEUSDT	PROMPTUSDT	PROMUSDT
PROSUSDT	PUFFERUSDT	PUMPUSDT	PUNDIXUSDT	PUNDUUSDT	PYRUSDT
PYTHUSDT	QIUSDT	QNTUSDT	QTUMUSDT	QUBICUSDT	QUICKUSDT
RADUSDT	RAREUSDT	RATSUSDT	RAYDIUMUSDT	RAYUSDT	RDNTUSDT
REDUSDT	REEFUSDT	RENDERUSDT	RENUSDT	REQUSDT	REXUSDT
REZUSDT	RFCUSDT	RIFSOLUSDT	RIFUSDT	RLCUSDT	RNDRUSDT
ROAMUSDT	RONINUSDT	RONUSDT	ROSEUSDT	RPLUSDT	RSRUSDT
RSS3USDT	RUNEUSDT	RVNUSDT	SAFEUSDT	SAGAUSDT	SANDUSDT
SATSUSDT	SCAUSDT	SCRTUSDT	SCRUSDT	SCUSDT	SDUSDT
SEIUSDT	SENDUSDT	SERAPHUSDT	SFPUSDT	SHELLUSDT	SHIB1000USDT
SIGNUSDT	SILLYUSDT	SIRENUSDT	SKLUSDT	SLERFUSDT	SLFUSDT
SLPUSDT	SNTUSDT	SNXUSDT	SOLAYERUSDT	SOLOUSDT	SOLUSD
SOLUSDT	SOLVUSDT	SONICUSDT	SPECUSDT	SPELLUSDT	SPXUSDT

SSVUSDT	STARLUSDT	STEEMUSDT	STGUSDT	STMXUSDT	STORJUSDT
STOUSDT	STPTUSDT	STRAXUSDT	STRKUSDT	STXUSDT	SUIUSD
SUIUSDT	SUNDOGUSDT	SUNUSDT	SUPERUSDT	SUSDT	SUSHIUSDT
SWARMSUSDT	SWEATUSDT	SWELLUSDT	SXPUSDT	SXTUSDT	SYNUSDT
SYSUSDT	TAIKOUSDT	TAIUSDT	TAOUSDT	TEST01SWELLUSDT	THETAUSDT
THEUSDT	TIAUSDT	TLMUSDT	TNSRUSDT	TOKENUSDT	TOMIUSDT
TOMOUSDT	TONUSD	TONUSDT	TOSHIUSDT	TRBUSDT	TROYUSDT
TRUMPUSDT	TRUUSDT	TRXUSDT	TSTBSCUSDT	TURBOUSDT	TUSDT
TUTUSDT	TWTUSDT	UMAUSDT	UNFIUSDT	UNIUSD	UNIUSDT
UROUSDT	USDCUSDT	USDEUSDT	USTCUSDT	USUALUSDT	UXLINKUSDT
VANAUSDT	VANRYUSDT	VELODROMEUSDT	VELOUSDT	VETUSDT	VGXUSDT
VICUSDT	VIDTUSDT	VINEUSDT	VINUUSDT	VIRTUALUSDT	VOXELUSDT
VRAUSDT	VRUSDT	VTHOUSDT	VVVUSDT	WALUSDT	WAVESUSDT
WAXPUSDT	WCTUSDT	WENUSDT	WHYUSDT	WIFUSD	WIFUSDT
WLDUSDT	WOOUSDT	WSMUSDT	WUSDT	XAIUSDT	XAUTUSDT
XCHUSDT	XCNUSDT	XDCUSDT	XECUSDT	XEMUSDT	XIONUSDT
XLMUSD	XLMUSDT	XMRUSDT	XNOUSDT	XRDUSDT	XRPUSD
XRPUSDT	XTERUSDT	XTZUSDT	XUSDT	XVGUSDT	XVSUSDT
YFIIUSDT	YFIUSDT	YGGUSDT	ZBCNUSDT	ZCXUSDT	ZECUSDT
ZENTUSDT	ZENUSDT	ZEREBROUSDT	ZETAUSDT	ZEUSUSDT	ZILUSDT
ZKFUSDT	ZKJUSDT	ZKUSDT	ZORAUSDT	ZRCUSDT	ZROUSDT
ZRXUSDT					

Example Applications

The Bybit Crypto Future Price dataset enables you to accurately design strategies for Crypto Futures with term structure. Examples include the following strategies:

- Horizontal/Diagonal arbitrage with the underlying Cryptocurrencies
- Trade Contango/Backwardation predictions
- Hedge for illiquid Cryptocurrencies

Classic Algorithm Example

The following example algorithm buys BTCUSDT perpetual future contract if the last day's close price was close to ask close

price than bid close price, sells short of that in opposite, through the Bybit exchange:

```C#
public class BybitCryptoFutureDataAlgorithm : QCAlgorithm
{
    public Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Set Account Currency to Tether, since USD and USDT will not auto-convert and USD cannot be used to trade
        SetAccountCurrency("USDT", 100000);
        // Bybit accepts both Cash and Margin account types, select the one you need for the best reality modeling.
        SetBrokerageModel(BrokerageName.Bybit, AccountType.Margin);

        // Requesting data, we only trade on BTCUSDT Future in Bybit exchange
        var cryptoFuture = AddCryptoFuture("BTCUSDT", Resolution.Daily);
        // perpetual futures does not have a filter function
        _symbol = cryptoFuture.Symbol;

        // Historical data
        var history = History(_symbol, 10, Resolution.Daily);
    }

    public override void OnData(Slice slice)
    {
        // Trade only based on updated price data
        if (!slice.QuoteBars.TryGetValue(_symbol, out var quote) || !slice.Bars.ContainsKey(_symbol))
        {
            return;
        }
        var price = slice.Bars[_symbol].Price;

        // Scalp-trade the bid-ask spread based on the supply-demand strength
        if (price - quote.Bid.Close > quote.Ask.Close - price)
        {
            SetHoldings(_symbol, -1m);
        }
        else
        {
            SetHoldings(_symbol, 1m);
        }
    }
}
```

Framework Algorithm Example

The following example algorithm hold a 100% long BTCUST future portfolio if the last day's close price was close to ask close price than bid close price, while hold short of that in opposite, through the Bybit exchange using the algorithm framework implementation:

```C#
public class BybitCryptoFutureDataAlgorithm : QCAlgorithm
{
    public Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Set Account Currency to Tether, since USD and USDT will not auto-convert and USD cannot be used to trade
        SetAccountCurrency("USDT", 100000);
        // Bybit accepts both Cash and Margin account types, select the one you need for the best reality modeling.
        SetBrokerageModel(BrokerageName.Bybit, AccountType.Margin);

        UniverseSettings.Resolution = Resolution.Daily;
        UniverseSettings.Leverage = 2;
        // We only trade on BTCUSDT Future in Bybit exchange
        var symbols = new []{
            QuantConnect.Symbol.Create("BTCUSDT", SecurityType.CryptoFuture, Market.Bybit)
        };
        AddUniverseSelection(new ManualUniverseSelectionModel(symbols));
        // Custom alpha model to emit insights based on the Crypto Future price data
```

```
        AddAlpha(new CryptoFutureAlphaModel());
        // Equally invest to evenly dissipate the capital concentration risk of inidividual crypto pair
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
        SetExecution(new ImmediateExecutionModel());
    }
}

public class CryptoFutureAlphaModel : AlphaModel
{
    private List<Symbol> _symbols = new();

    public override List<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = new List<Insight>();

        foreach (var symbol in _symbols)
        {
            // Trade only based on updated price data
            if (!slice.Bars.ContainsKey(symbol) || !slice.QuoteBars.TryGetValue(symbol, out var quote))
            {
                continue;
            }
            var price = slice.Bars[symbol].Price;

            // Scalp-trade the bid-ask spread based on the supply-demand strength
            if (price - quote.Bid.Close > quote.Ask.Close - price)
            {
                insights.Add(
                    Insight.Price(symbol, TimeSpan.FromDays(1), InsightDirection.Down)
                );
            }
            else
            {
                insights.Add(
                    Insight.Price(symbol, TimeSpan.FromDays(1), InsightDirection.Up)
                );
            }
        }

        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            var symbol = security.Symbol;
            _symbols.Add(symbol);

            // Historical data
            var history = algorithm.History(symbol, 10, Resolution.Daily);
        }

        foreach (var security in changes.RemovedSecurities)
        {
            _symbols.Remove(security.Symbol);
        }
    }
}
```

Data Point Attributes

The Bybit Crypto Future Price dataset provides `TradeBar` , `QuoteBar` , and `Tick` objects.

TradeBar Attributes

`TradeBar` objects have the following attributes:

QuoteBar Attributes

`QuoteBar` objects have the following attributes:

Tick Attributes

`Tick` objects have the following attributes:

CoinAPI

Bybit Crypto Price Data

Introduction

The Bybit Crypto Price Data by CoinAPI is for Cryptocurrency price and volume data points. The data covers 956 Cryptocurrency pairs, starts in July 2017, and is delivered on any frequency from tick to daily. This dataset is created by monitoring the trading activity on Bybit.

For more information about the Bybit Crypto Price Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

CoinAPI was founded by Artur Pietrzyk in 2016 with the goal of providing real-time and historical cryptocurrency market data, collected from hundreds of exchanges. CoinAPI provides access to Cryptocurrencies for traders, market makers, and developers building third-party applications.

Getting Started

The following snippet demonstrates how to request data from the Bybit Crypto Price dataset:

```C#
// Bybit accepts both Cash and Margin account types only.
SetBrokerageModel(BrokerageName.Bybit, AccountType.Cash);
//SetBrokerageModel(BrokerageName.Bybit, AccountType.Margin);

_symbol = AddCrypto("BTCUSDT", Resolution.Minute, Market.Bybit).Symbol;

_universe = AddUniverse(CryptoUniverse.Bybit(UniverseSelectionFilter));
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	April 2022
Asset Coverage	956 Currency Pairs
Data Density	Dense
Resolution	Tick, Second, Minute, Hourly, & Daily
Timezone	UTC
Market Hours	Always Open

Requesting Data

To add Bybit Crypto Price data to your algorithm, call the `AddCrypto` method. Save a reference to the Crypto `Symbol` so you can access the data later in your algorithm.

```
C#
public class CoinAPIDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2020, 6, 1);
        SetEndDate(2021, 6, 1);

        // Set Account Currency to Tether
        SetAccountCurrency("USDT", 100000);

        // Bybit accepts both Cash and Margin account types.
        SetBrokerageModel(BrokerageName.Bybit, AccountType.Margin);

        _symbol = AddCrypto("BTCUSDT", Resolution.Minute, Market.Bybit).Symbol;
    }
}
```

For more information about creating Crypto subscriptions, see Requesting Data .

Accessing Data

To get the current Bybit Crypto Price data, index the `Bars` , `QuoteBars` , or `Ticks` properties of the current `Slice` with the Crypto `Symbol` . `Slice` objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.Bars.ContainsKey(_symbol))
    {
        var tradeBar = slice.Bars[_symbol];
        Log($"{_symbol} price at {slice.Time}: {tradeBar.Close}");
    }

    if (slice.QuoteBars.ContainsKey(_symbol))
    {
        var quoteBar = slice.QuoteBars[_symbol];
        Log($"{_symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    if (slice.Ticks.ContainsKey(_symbol))
    {
        var ticks = slice.Ticks[_symbol];
        foreach (var tick in ticks)
        {
            Log($"{_symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

You can also iterate through all of the data objects in the current `Slice` .

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Bars)
    {
        var symbol = kvp.Key;
        var tradeBar = kvp.Value;
        Log($"{symbol} price at {slice.Time}: {tradeBar.Close}");
    }

    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        Log($"{symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    foreach (var kvp in slice.Ticks)
    {
        var symbol = kvp.Key;
        var ticks = kvp.Value;
        foreach (var tick in ticks)
        {
            Log($"{symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

For more information about accessing Crypto data, see Handling Data .

Historical Data

To get historical Bybit Crypto Price data, call the `History` method with the Crypto `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
// TradeBar objects
var historyTradeBars = History(_symbol, 100, Resolution.Daily);

// QuoteBar objects
var historyQuoteBars = History<QuoteBar>(_symbol, 100, Resolution.Minute);

// Tick objects
var historyTicks = History<Tick>(_symbol, TimeSpan.FromSeconds(10), Resolution.Tick);
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of Bybit Crypto pairs, call the `AddUniverse` method with a `CryptoUniverse` object. A Crypto universe uses a selection function to select Crypto pairs based on their OHLCV and dollar volume of the previous day as of midnight Coordinated Universal Time (UTC).

```C#
using QuantConnect.Data.UniverseSelection;

public override void Initialize()
{
    SetBrokerageModel(BrokerageName.Bybit, AccountType.Margin);
    UniverseSettings.Asynchronous = true;
    _universe = AddUniverse(CryptoUniverse.Bybit(UniverseSelectionFilter));
}

private IEnumerable<Symbol> UniverseSelectionFilter(IEnumerable<CryptoUniverse> universeDay)
{
    return from c in universeDay
           where c.Volume >= 100m && c.VolumeInUsd > 10000m
           select c.Symbol;
}
```

For more information about universe settings, see Settings .

Universe History

You can get historical universe data in an algorithm and in the Research Environment.

Historical Universe Data in Algorithms

To get historical universe data in an algorithm, call the `History` method with the `Universe` object, and the lookback period. If there is no data in the period you request, the history result is empty.

```C#
var history = History(_universe, 30, Resolution.Daily);
foreach (var universeDay in history)
{
    foreach (CryptoUniverse universeItem in universeDay)
    {
        Log($"{universeItem.Symbol} price at {universeItem.EndTime}: {universeItem.Close}");
    }
}
```

Historical Universe Data in Research

To get historical universe data in research, call the `UniverseHistory` method with the `Universe` object, and the lookback period. The `UniverseHistory` returns the filtered universe. If there is no data in the period you request, the history result is empty.

```C#
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var universeDay in universeHistory)
{
    foreach (CryptoUniverse universeItem in universeDay)
    {
        Console.WriteLine($"{universeItem.Symbol} price at {universeItem.EndTime}: {universeItem.Close}");
    }
}
```

You can call the `History` method in Research.

Remove Subscriptions

To unsubscribe from a Crypto pair that you added with the `AddCrypto` method, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_symbol);
```

The `RemoveSecurity` method cancels your open orders for the security and liquidates your holdings in the virtual pair .

Supported Assets

The following table shows the available Cryptocurrency pairs:

Pairs Available (956)					
1INCHADA	1INCHETH	1INCHUSDE	1INCHUSDT	1SOLUSDT	3PUSDT
5IREUSDC	5IREUSDT	A8USDT	AARKUSDT	AAVEACH	AAVEUSDC
AAVEUSDT	ACAUSDC	ACAUSDT	ACHUSDC	ACHUSDT	ACMUSDT
ACSACH	ACSMNT	ACSUSDC	ACSUSDT	ADA2LUSDT	ADA2SUSDT
ADABTC	ADAETH	ADAEUR	ADAMNT	ADAUSDC	ADAUSDT
AEGUSDT	AEROUSDT	AEVOUSDT	AFCUSDT	AFGUSDT	AGIUSDT
AGIXUSDT	AGLAUSDT	AGLDUSDT	AI16ZUSDT	AIOZUSDT	AIXBTUSDT
AKIUSDT	ALCHUSDT	ALGOBTC	ALGOUSDT	ALTUSDT	AMIUSDT
ANIMEUSDT	ANKRUSDT	APE2LUSDT	APE2SUSDT	APEUSDC	APEUSDT
APEXUSDC	APEXUSDT	APPUSDT	APRSUSDT	APTRUSDT	APTUSDC
APTUSDT	ARBUSDC	ARBUSDT	ARKMUSDT	ARTYUSDT	ARUSDT
ATHUSDT	ATOM2LUSDT	ATOM2SUSDT	ATOMUSDC	ATOMUSDT	AUDETH
AUDUSDT	AURORAUSDT	AURYUSDT	AVAILUSDT	AVAUSDT	AVAX2LUSDT
AVAX2SUSDT	AVAXEUR	AVAXUSDC	AVAXUSDT	AVLUSDT	AXLUSDT
AXSUSDT	AZYUSDT	B3TRUSDC	B3TRUSDT	B3USDT	BABY1USDT
BABYDOGEBTC	BABYDOGEETH	BABYDOGEUSDT	BANUSDT	BARUSDT	BATBTC
BATETH	BATUSDT	BBDETH	BBDUSDC	BBDUSDT	BBLUSDT
BBQUSDT	BBSOLSOL	BBSOLUSDC	BBSOLUSDT	BBUSDT	BCHUSDC
BCHUSDT	BCUTUSDT	BEAMUSDT	BELUSDT	BENUSDT	BERAUSDT
BICOUSDT	BLASTUSDT	BLOCKUSDT	BLURUSDT	BMTUSDT	BNBETH
BNBUSDC	BNBUSDT	BNTUSDT	BOBAUSDT	BOBUSDC	BOBUSDT
BOMEUSDC	BOMEUSDT	BONKUSDC	BONKUSDT	BONUSETH	BONUSEUR
BONUSUSDC	BONUSUSDT	BRAWLUSDT	BRETTUSDC	BRETTUSDT	BRLETH
BRUSDT	BRZUSDT	BTC2SUSDT	BTC3LUSDC	BTC3LUSDT	BTC3SUSDT

BTCBETH	BTCBRL	BTCBRZ	BTCCUSDT	BTCDAI	BTCETH
BTCEUR	BTCPLN	BTCTRY	BTCUSDC	BTCUSDE	BTCUSDQ
BTCUSDR	BTCUSDT	BTGUSDT	BTTBIT	BTTUSDT	BUBBLEUSDT
BUSDUSDT	C98USDT	CAKEUSDT	CANDYUSDT	CAPOUSDT	CAPSUSDT
CARVUSDT	CATBNBUSDT	CATIEUR	CATIUSDC	CATIUSDT	CATSUSDT
CATUSDT	CBKUSDT	CBXUSDT	CELOUSDT	CELUSDT	CGPTUSDT
CHILLGUYUSDT	CHRPUSDT	CHZBIT	CHZEUR	CHZUSDC	CHZUSDT
CITYUSDT	CLOUDUSDT	CMETHUSDT	CMPUSDT	COMPBIT	COMPUSDT
COMUSDT	COOKIEUSDT	COOKUSDT	COPBIT	COPETH	COPUSDT
COQUSDT	COREUSDC	COREUSDT	CORNUSDT	COTUSDT	COUSDT
CPOOLUSDT	CRDSUSDT	CRVBTC	CRVETH	CRVUSDT	CSPRUSDT
CTAUSDT	CTCETH	CTCUSDT	CTTUSDT	CULTUSDT	CUSDUSDT
CWARUSDT	CYBERUSDT	DAIADA	DAIBIT	DAIETH	DAIUSDC
DAIUSDT	DAPPUSDT	DBRUSDT	DCRBIT	DCRUSDT	DECHATUSDT
DEEPUSDT	DEFIUSDT	DEFYUSDT	DEGENUSDT	DEVTUSDT	DFIUSDT
DGBUSDT	DIAMUSDT	DICEUSDT	DLCUSDT	DMAILUSDT	DOGE2LUSDT
DOGE2SUSDT	DOGEETH	DOGEEUR	DOGEUSDC	DOGEUSDT	DOGSEUR
DOGSUSDC	DOGSUSDT	DOLOUSDT	DOMEUSDT	DOODUSDT	DOP1USDT
DOT3LUSDT	DOT3SUSDT	DOTBTC	DOTETH	DOTUSDC	DOTUSDT
DPXUSDT	DRIFTUSDT	DSRUNUSDT	DUELUSDT	DYDXUSDC	DYDXUSDT
DYMUSDT	DZDUSDT	DZOOUSDT	ECOXUSDT	EGLDUSDC	EGLDUSDT
EGOUSDT	EGP1USDT	EIGENUSDT	ELDAUSDT	ELIXUSDT	ELTUSDT
ELXUSDT	ENAEUR	ENAUSDT	ENJUSDT	ENSUSDT	EOS2LUSDT
EOS2SUSDT	EOS3LUSDT	EOS3SUSDT	EOSBTC	EOSETH	EOSMNT
EOSTUSD	EOSUSDC	EOSUSDT	EPTUSDT	ERTHAUSDT	ESEUSDT
ETC2LUSDT	ETC2SUSDT	ETCETH	ETCUSDT	ETH3LUSDT	ETH3SUSDT
ETHBRL	ETHBTC	ETHDAI	ETHEUR	ETHFIUSDT	ETHPLN
ETHTRY	ETHUSDC	ETHUSDE	ETHUSDT	ETHWUSDT	EUROCUSDT

EVERUSDT	EVERYUSDT	EXVGUSDT	FAMEUSDT	FARUSDT	FBUSDT
FCDUSDT	FETUSDC	FETUSDT	FHEUSDT	FIDAUSDT	FILUSDC
FILUSDT	FIREUSDT	FITFIUSDT	FLIPUSDT	FLOCKUSDT	FLOKIUSDC
FLOKIUSDT	FLOWBIT	FLOWUSDT	FLRUSDT	FLTUSDT	FLUIDUSDT
FMBUSDT	FMCUSDT	FONUSDT	FORTUSDT	FOXYUSDT	FTM2LUSDT
FTM2SUSDT	FTMEUR	FTMUSDC	FTMUSDT	FTTUSDC	FTTUSDT
FUELUSDT	FUSDT	FXSBIT	FXSUSDT	G3USDT	G7USDT
GALAUSDT	GALAXISUSDT	GALFTUSDT	GALUSDT	GAMEUSDT	GCAKEUSDT
GENEUSDT	GGMUSDT	GGUSDT	GLMRUSDT	GMBIT	GMRXUSDT
GMT2LUSDT	GMT2SUSDT	GMTUSDC	GMTUSDT	GMUSDC	GMUSDT
GMXBIT	GMXETH	GMXUSDT	GNSUSDT	GOATUSDT	GODSUSDT
GPSUSDT	GPTUSDT	GRAPEUSDT	GRASSUSDT	GRTBIT	GRTUSDT
GSTSUSDT	GSTUSDT	GSWIFTUSDT	GTAIUSDT	GUMMYUSDT	GUSDT
HAEDALUSDT	HATUSDT	HBARUSDT	HEROUSDT	HFTUSDC	HFTUSDT
HLGUSDT	HMSTRUSDC	HMSTRUSDT	HNTBIT	HNTUSDT	HONUSDT
HOOKUSDT	HOTUSDT	HPOS10IUSDT	HTETH	HTXUSDT	HVHUSDT
HYPERUSDT	ICPUSDC	ICPUSDT	ICXUSDT	IDUSDT	IMXUSDT
INITUSDT	INJUSDC	INJUSDT	INSPUSDT	INTERUSDT	INTXUSDT
IOUSDT	IPUSDT	IRLUSDT	IZIUSDT	JASMYUSDC	JASMYUSDT
JEFFUSDT	JSTUSDT	JTOUSDT	JUPUSDT	JUSDT	JUVUSDT
KAIAUSDT	KARATEUSDT	KASTAUSDT	KASUSDC	KASUSDT	KAVAUSDT
KCALUSDT	KCSUSDT	KDAUSDT	KILOUSDT	KLAYUSDT	KMABIT
KMAUSDT	KMNOUSDT	KMONUSDT	KOKUSDT	KONUSDT	KRLUSDT
KROUSDT	KSMUSDT	KUBUSDT	KUNCIUSDT	L3USDT	LADYSUSDT
LAIUSDT	LAVAUSDT	LAYERUSDT	LDOUSDC	LDOUSDT	LENDSUSDT
LEVERUSDT	LFTUSDT	LFWUSDT	LGXUSDT	LINGUSDT	LINK2LUSDT
LINK2SUSDT	LINKEUR	LINKUSDC	LINKUSDT	LISUSDT	LLUSDT
LMWRUSDT	LNK2SUSDT	LOOKSUSDT	LRCUSDT	LTC2LUSDT	LTC2SUSDT

LTCBTC	LTCEUR	LTCUSDC	LTCUSDT	LUCEUSDT	LUNAIUSDT
LUNAUSDT	LUNCUSDC	LUNCUSDT	MAGICUSDT	MAJORUSDT	MAKUSDT
MANABTC	MANAUSDC	MANAUSDT	MANTAUSDT	MASAUSDT	MASKUSDT
MATIC2LUSDT	MATIC2SUSDT	MATICBTC	MATICUSDC	MATICUSDT	MAVIAUSDT
MBOXUSDT	MBSUSDT	MBXUSDT	MCGUSDT	MCRTUSDT	MCTUSDT
MDAOUSDT	MEEETH	MEEUSDC	MEEUSDT	MELOSUSDT	MEMEFIUSDT
MEMEUSDT	MERLUSDT	METHETH	METHUSDT	MEUSDT	MEWUSDC
MEWUSDT	MFERUSDT	MIBRUSDT	MILKUSDT	MINABTC	MINAETH
MINAMNT	MINAUSDC	MINAUSDT	MINUUSDT	MIXUSDT	MKRUSDT
MLKUSDT	MMCUSDT	MNRYUSDT	MNTBTC	MNTETH	MNTUSDC
MNTUSDT	MNZUSDT	MOCAUSDT	MODEUSDT	MOGUSDT	MOJOUSDT
MONUSDT	MORPHOUSDT	MOVEUSDT	MOVEZUSDT	MOVRUSDT	MOZUSDT
MPLXUSDT	MSTARUSDT	MTCUSDT	MTKUSDT	MUSDUSDT	MVLUSDT
MVUSDT	MXMUSDT	MXUSDT	MYRIAUSDT	MYROUSDT	N3USDT
NAKAUSDT	NAVXUSDT	NEAREUR	NEARUSDC	NEARUSDT	NEIROCTOUSDT
NEIROUSDT	NEONUSDT	NESSUSDT	NETHETH	NETHUSDT	NEXOUSDT
NEXTUSDT	NFTUSDT	NGLUSDT	NIBIUSDT	NLKUSDT	NOTUSDC
NOTUSDT	NRNUSDT	NSUSDT	NUTSUSDT	NUUSDT	NYANUSDT
NYMUSDT	OASUSDT	OBOLUSDT	OBTUSDT	OBXUSDT	ODOSUSDT
OIKUSDT	OKGUSDT	OLUSDT	OMGUSDT	OMNICATUSDC	OMNICATUSDT
OMNIUSDC	OMNIUSDT	OMNUSDT	OMUSDT	ONDOEUR	ONDOUSDC
ONDOUSDT	ONEUSDT	OPUSDC	OPUSDT	ORDERUSDT	ORDIUSDT
ORTUSDT	PAALUSDT	PARAMUSDT	PARTIUSDT	PAWSUSDT	PAXGUSDT
PBUXUSDT	PELLUSDT	PENDLEUSDT	PENGUUSDT	PENUSDT	PEOPLEUSDT
PEPE2USDC	PEPE2USDT	PEPEEUR	PEPEUSDC	PEPEUSDT	PERPUSDT
PHPUSDT	PINEYEUSDT	PIPUSDT	PIRATEUSDT	PIXFIUSDT	PLANETUSDT
PLAYUSDT	PLTUSDT	PLUMEUSDT	PLYUSDT	PNUTUSDT	POKTUSDT
POLUSDT	PONKEUSDT	POPCATUSDT	PORT3USDT	PORTALUSDT	POSIUSDC

POSIUSDT	PPTUSDT	PRCLUSDT	PRIMALUSDT	PRIMEUSDT	PSGUSDT
PSPUSDT	PSTAKEUSDT	PTCUSDT	PTUUSDT	PUFFERUSDT	PUFFUSDT
PUMLXUSDT	PUMPUSDT	PURSEUSDT	PYTHUSDT	PYUSDUSDT	QMALLUSDT
QNTUSDT	QORPOUSDT	QTUMUSDT	RACAUSDT	RAINUSDT	RATSUSDT
RDNTUSDT	REALUSDT	REDUSDT	RENDERUSDT	RENUSDT	RLTMUSDT
RNDRUSDT	ROAMUSDT	RONDUSDT	ROOTUSDT	ROSEUSDT	ROUTEUSDT
RPKUSDT	RPLUSDT	RSS3USDT	RUBUSDT	RUBYUSDT	RUNBIT
RUNBTC	RUNEUSDT	RUNUSDT	RVNUSDT	SAFEUSDT	SAILUSDT
SAITAMAUSDT	SALDUSDT	SAND2LUSDT	SAND2SUSDT	SANDBTC	SANDUSDC
SANDUSDT	SAROSUSDT	SATSUSDT	SCAUSDT	SCRTUSDT	SCRUSDT
SCUSDT	SDAGLD	SDBIT	SDUSDT	SEIEUR	SEILORUSDT
SEIUSDC	SEIUSDT	SENDUSDT	SEORUSDT	SERAPHUSDT	SFUNDUSDT
SHARKUSDT	SHIBEUR	SHIBUSDC	SHIBUSDT	SHILLUSDT	SHRAPUSDT
SIDUSUSDT	SIGNUSDT	SISUSDT	SLGUSDC	SLGUSDT	SLPUSDT
SMILEUSDT	SNXUSDT	SOCIALUSDT	SOLBRL	SOLBTC	SOLETH
SOLEUR	SOLOUSDT	SOLUSDC	SOLUSDE	SOLUSDT	SOLVUSDT
SOLXRP	SONBIT	SONICUSDT	SONUSDT	SOSOUSDT	SOSUSDT
SPARTAUSDT	SPECUSDT	SPELLUSDT	SPXUSDT	SQDUSDT	SQRUSDT
SQTUSDT	SRMUSDT	SSVUSDT	STARUSDT	STATUSDT	STETHEUR
STETHUSDT	STGUSDT	STOPUSDT	STREAMUSDT	STRKUSDC	STRKUSDT
STRMUSDT	STXUSDT	SUIAUSDT	SUIUSDC	SUIUSDT	SUNDOGUSDT
SUNUSDT	SUPRAUSDT	SUSDT	SUSHIUSDT	SVLUSDT	SWEATUSDT
SWELLUSDC	SWELLUSDT	SXTUSDT	SYNRUSDT	TAIKOUSDT	TAIUSDT
TAMAUSDT	TAPUSDT	TAVAUSDT	TELUSDT	TENETUSDT	TFUELUSDT
THBEUR	THBUSDC	THBUSDT	THETAUSDT	THNUSDT	THRUSTUSDT
TIAUSDC	TIAUSDT	TIMEUSDT	TNSRUSDT	TOKENUSDT	TOMIUSDT
TOMSUSDT	TONEUR	TONUSDC	TONUSDT	TOSHIUSDT	TRCUSDT
TRIBEUSDT	TRUMPUSDC	TRUMPUSDT	TRVLUSDT	TRXUSDC	TRXUSDT

TSTUSDT	TURBOSUSDT	TUSDT	TUSDUSDT	TVKUSDT	TWTUSDT
ULTIUSDT	UMAUSDT	UNIUSDC	UNIUSDT	USDCBRL	USDCEUR
USDCUSDT	USDCZEC	USDDUSDT	USDEEUR	USDEUSDC	USDEUSDT
USDQUSDT	USDRUSDT	USDTAUD	USDTBRL	USDTBRZ	USDTBUSDT
USDTETH	USDTEUR	USDTPLN	USDTTRY	USDTUSDE	USDTZEC
USDUSDT	USDYUSDT	USTCUSDT	UXLINKUSDT	UYUUSDT	VANAUSDT
VANRYUSDT	VEGAUSDT	VELARUSDT	VELAUSDT	VELOUSDT	VENOMUSDT
VETUSDT	VEXTUSDT	VICUSDT	VINUUSDT	VIRTUALUSDT	VPADUSDT
VPRUSDT	VRAUSDT	VRTXUSDT	VTHOUSDT	VVUSDT	VVVUSDT
WALUSDT	WAVESUSDT	WAXPUSDT	WBTCBTC	WBTCUSDT	WCTUSDT
WEETHETH	WELLUSDT	WEMIXUSDT	WENUSDT	WIFEUR	WIFUSDC
WIFUSDT	WLDEUR	WLDUSDC	WLDUSDT	WLKNUSDT	WOOUSDT
WUSDT	WWYUSDT	XAIUSDT	XARUSDT	XAUTUSDT	XAVAUSDT
XCADUSDT	XDCUSDT	XECETH	XECUSDC	XECUSDT	XEMUSDT
XETAUSDT	XIONUSDT	XLMBTC	XLMUSDC	XLMUSDT	XRP3LETH
XRP3LUSDT	XRP3SUSDT	XRPBTC	XRPETH	XRPEUR	XRPUSDC
XRPUSDT	XTERUSDT	XTZUSDT	XUSDT	XWGUSDT	XYMUSDT
XZKUSDT	YFIBIT	YFIUSDT	ZAMUSDT	ZAPUSDT	ZARUSDT
ZBCBIT	ZBCUSDT	ZECBTC	ZECETH	ZECTFUEL	ZECUSDT
ZENBTC	ZENDUSDT	ZENETH	ZENTUSDT	ZENUSDC	ZENUSDT
ZEREBROUSDT	ZEROUSDT	ZETAUSDT	ZEXUSDT	ZIGUSDT	ZILUSDT
ZKFUSDT	ZKJETH	ZKJUSDT	ZKLUSDT	ZKUSDC	ZKUSDT
ZORAUSDT	ZRCUSDT	ZROUSDC	ZROUSDT	ZRXACS	ZRXETH
ZRXUSDT	ZTXUSDT				

Example Applications

The Bybit Crypto Price dataset enables you to accurately design strategies for Cryptocurrencies. Examples include the following strategies:

- Buy and hold
- Trading Cryptocurrency volatility and price action
- Allocating a small portion of your portfolio to Cryptocurrencies to hedge against inflation

Classic Algorithm Example

The following example algorithm buys and holds Bitcoin through the Bybit exchange:

```
from AlgorithmImports import * class CoinAPIDataAlgorithm(QCAlgorithm): def initialize(self) -> None: self.set_start_date(2024, 9, 1) self.set_end_date(2024, 12, 31) # Set the account currency to USDT, since you can't trade with # USD and the algorithm doesn't automatically convert USD & USDT # holdings. self.set_account_currency("USDT", 100_000) self.universe_settings.asynchronous = True # Bybit accepts both Cash and Margin account types. Select the # one you need for the best reality modeling. self.set_brokerage_model(BrokerageName.BYBIT, AccountType.MARGIN) # Add BTCUSDT data. self._btc = self.add_crypto("BTCUSDT", Resolution.MINUTE, Market.BYBIT) # Historical data history = self.history(self._btc.symbol, 30, Resolution.DAILY) # Add a Crypto universe that selects trading pairs on Bybit. universe = self.add_universe(CryptoUniverse.bybit(self._select_assets)) # Historical Universe data universe_history = self.history(universe, 30, Resolution.DAILY) for (universe_symbol, time), universe_day in universe_history.items(): for universe_item in universe_day: symbol = universe_item.symbol price = universe_item.close def _select_assets(self, universe_day): # Filter for materially traded Crypto pairs with significant # size and dollar volume, assuming higher capital flow in for # higher return. return [universe_item.symbol for universe_item in universe_day if universe_item.volume >= 100 and universe_item.volume_in_usd > 10_000_000] def on_data(self, slice: Slice) -> None: # Speculate-invest all available free cash on BTCUSDT, repecting # the order quantity restrictions to avoid invalid orders. if self.portfolio.cash_book['BTC'].amount != 0: return # Define a buffer so that we can scale down the order size, # avoiding trading errors. buffer = max( BybitFeeModel.TAKER_NON_VIP_FEE, self.settings.free_portfolio_value_percentage ) quantity = ( self.portfolio.cash_book['USDT'].amount / self._btc.ask_price * (1-buffer) ) # Check if the quantity exceeds the minimum order size. On # Bybit, the minimum order size is measured in the base # currency (BTC in this case). if abs(quantity) < self._btc.symbol_properties.minimum_order_size: return self.market_order(self._btc, quantity)
```

```
public class CoinAPIDataAlgorithm : QCAlgorithm { private Crypto _btc; public override void Initialize() { SetStartDate(2024, 9, 1); SetEndDate(2024, 12, 31); // Set the account currency to USDT, since you can't trade with // USD and the algorithm doesn't automatically convert USD & USDT // holdings. SetAccountCurrency("USDT", 100000); UniverseSettings.Asynchronous = true; // Bybit accepts both Cash and Margin account types. Select the // one you need for the best reality modeling. SetBrokerageModel(BrokerageName.Bybit, AccountType.Margin); // Add BTCUSDT data. _btc = AddCrypto("BTCUSDT", Resolution.Minute, Market.Bybit); // Historical data var history = History(_btc.Symbol, 30, Resolution.Daily); // Add a Crypto universe that selects trading pairs on Bybit. var universe = AddUniverse(CryptoUniverse.Bybit(SelectAssets)); // Historical Universe data var universeHistory = History(universe, 30, Resolution.Daily); foreach (var universeDay in universeHistory) { foreach (CryptoUniverse universeItem in universeDay) { var symbol = universeItem.Symbol; var price = universeItem.Close; } } } private IEnumerable<Symbol> SelectAssets(IEnumerable<CryptoUniverse> universeDay) { // Filter for materially traded Crypto pairs with significant // size and dollar volume, assuming higher capital flow in for // higher return. return from universeItem in universeDay where universeItem.Volume >= 100m && universeItem.VolumeInUsd > 10000000m select universeItem.Symbol; } public override void OnData(Slice slice) { // Speculate-invest all available free cash on BTCUSDT, repecting // the order quantity restrictions to avoid invalid orders. if (Portfolio.CashBook["BTC"].Amount != 0) { return; } // Define a buffer so that we can scale down the order size, // avoiding trading errors. var buffer = Math.Max(BybitFeeModel.TakerNonVIPFee, Settings.FreePortfolioValuePercentage); var quantity = Portfolio.CashBook["USDT"].Amount / _btc.AskPrice * (1-buffer); // Check if the quantity exceeds the minimum order size. On // Bybit, the minimum order size is measured in the base // currency (BTC in this case). if (Math.Abs(quantity) < _btc.SymbolProperties.MinimumOrderSize) { return; } MarketOrder(_btc, quantity); } }
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of Crypto pairs on the Bybit exchange and then forms a equal-weighted portfolio of all the pairs in the universe:

```C#
public class CoinAPIDataAlgorithm : QCAlgorithm
{
    private List<string> _marketPairs;
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        Settings.FreePortfolioValuePercentage = 0.05m;

        // Set Account Currency to Tether, since USD and USDT will not auto-convert and USD cannot be used to trade
        SetAccountCurrency("USDT", 100000);
        // Bybit accepts both Cash and Margin account types, select the one you need for the best reality modeling.
        SetBrokerageModel(BrokerageName.Bybit, AccountType.Margin);

        // Warm up the security with the last known price to avoid conversion error
        Settings.SeedInitialPrices = true;
        UniverseSettings.Asynchronous = true;
        UniverseSettings.Resolution = Resolution.Minute;
        // Add Crypto Universe Selection that select crypto pairs in Bybit exchange
        AddUniverse(CryptoUniverse.Bybit(UniverseSelectionFilter));
        // Get the pairs on Bybit that have USDT as the quote currency.
        _marketPairs = SymbolPropertiesDatabase.GetSymbolPropertiesList(Market.Bybit)
            .Where(x => x.Value.QuoteCurrency == AccountCurrency)
            .Select(x => x.Key.Symbol)
            .ToList();

        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromMinutes(20), 0.025,
null));
        // Equally invest to evenly dissipate the capital concentration risk of inidividual crypto pair
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }

    private IEnumerable<Symbol> UniverseSelectionFilter(IEnumerable<CryptoUniverse> universeDay)
    {
        // Select pairs where the quote currency is USDT.
        return universeDay.Where(c => _marketPairs.Contains(c.Symbol.Value))
            // Filter for materially traded crypto pairs with significant size and dollar volume, assuming higher
capital flow in for higher return
            .OrderBy(c => c.VolumeInUsd)
            .TakeLast(3)
            .Select(c => c.Symbol);
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach(var security in changes.AddedSecurities)
        {
            // Historical data
            var history = History(security.Symbol, 30, Resolution.Daily);
            Debug($"We got {history.Count()} items from our history request");
        }
    }
}
```

Research Example

The following example lists crypto-currency pairs with the greatest dollar volume in the Bybit exchange:

```C#
var qb = new QuantBook();

// Add Cryptocurrency pair
var symbol = qb.AddCrypto("BTCUSDT", market:Market.Bybit).Symbol;

// Historical data
var history = qb.History(symbol, 30, Resolution.Daily);
foreach (var bar in history)
{
    Console.WriteLine($"{bar.EndTime} {bar}");
}

// Add Crypto Universe Selection
IEnumerable<Symbol> UniverseSelectionFilter(IEnumerable<CryptoUniverse> universeDay)
{
    return universeDay
        .Where(x => x.VolumeInUsd != null && x.VolumeInUsd > 10000m)
        .OrderByDescending(x => x.VolumeInUsd)
        .Take(5)
        .Select(x => x.Symbol);
}
var universe = qb.AddUniverse(CryptoUniverse.Bybit(UniverseSelectionFilter));

// Historical Universe data
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var universeDay in universeHistory)
{
    Console.WriteLine($"=== {universeDay.First().EndTime} ===");
    foreach (CryptoUniverse universeItem in universeDay.OrderByDescending(x => x.VolumeInUsd))
    {
        Console.WriteLine($"{universeItem.Symbol}: {universeItem.VolumeInUsd}");
    }
}
```

Data Point Attributes

The Bybit Crypto Price dataset provides `TradeBar` , `QuoteBar` , `Tick` , and `CryptoUniverse` objects.

TradeBar Attributes

`TradeBar` objects have the following attributes:

QuoteBar Attributes

`QuoteBar` objects have the following attributes:

Tick Attributes

`Tick` objects have the following attributes:

CryptoUniverse Attributes

`CryptoUniverse` objects have the following attributes:

CoinAPI

Coinbase Crypto Price Data

Introduction

The Coinbase Crypto Price Data by CoinAPI provides Cryptocurrency price and volume data points. The data covers 970 Cryptocurrency pairs, starts in January 2015, and is delivered on any frequency from tick to daily. This dataset is created by monitoring the trading activity on Coinbase.

For more information about the Coinbase Crypto Price Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

CoinAPI was founded by Artur Pietrzyk in 2016 with the goal of providing real-time and historical cryptocurrency market data, collected from hundreds of exchanges. CoinAPI provides access to Cryptocurrencies for traders, market makers, and developers building third-party applications.

Getting Started

The following snippet demonstrates how to request data from the Coinbase Crypto Price dataset:

```C#
// Coinbase only accepts Cash account type
SetBrokerageModel(BrokerageName.Coinbase, AccountType.Cash);

_symbol = AddCrypto("BTCUSD", Resolution.Minute, Market.Coinbase).Symbol;

_universe = AddUniverse(CryptoUniverse.Coinbase(UniverseSelectionFilter));
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2015
Asset Coverage	970 Currency Pairs
Data Density	Dense
Resolution	Tick, Second, Minute, Hourly, & Daily
Timezone	UTC
Market Hours	Always Open

Requesting Data

To add Coinbase Crypto Price data to your algorithm, call the `AddCrypto` method. Save a reference to the Crypto `Symbol` so you can access the data later in your algorithm.

```
C#
public class CoinAPIDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2020, 6, 1);
        SetEndDate(2021, 6, 1);
        SetCash(100000);

        // Coinbase accepts Cash account type only, AccountType.Margin will result in an exception.
        SetBrokerageModel(BrokerageName.Coinbase, AccountType.Cash);

        _symbol = AddCrypto("BTCUSD", Resolution.Minute, Market.Coinbase).Symbol;
    }
}
```

For more information about creating Crypto subscriptions, see Requesting Data .

Accessing Data

To get the current Coinbase Crypto Price data, index the `Bars` , `QuoteBars` , or `Ticks` properties of the current `Slice` with the Crypto `Symbol` . `Slice` objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.Bars.ContainsKey(_symbol))
    {
        var tradeBar = slice.Bars[_symbol];
        Log($"{_symbol} price at {slice.Time}: {tradeBar.Close}");
    }

    if (slice.QuoteBars.ContainsKey(_symbol))
    {
        var quoteBar = slice.QuoteBars[_symbol];
        Log($"{_symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    if (slice.Ticks.ContainsKey(_symbol))
    {
        var ticks = slice.Ticks[_symbol];
        foreach (var tick in ticks)
        {
            Log($"{_symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

You can also iterate through all of the data objects in the current `Slice` .

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Bars)
    {
        var symbol = kvp.Key;
        var tradeBar = kvp.Value;
        Log($"{symbol} price at {slice.Time}: {tradeBar.Close}");
    }

    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        Log($"{symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    foreach (var kvp in slice.Ticks)
    {
        var symbol = kvp.Key;
        var ticks = kvp.Value;
        foreach (var tick in ticks)
        {
            Log($"{symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

For more information about accessing Crypto data, see Handling Data .

Historical Data

To get historical Coinbase Crypto Price data, call the `History` method with the Crypto `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
// TradeBar objects
var historyTradeBars = History(_symbol, 100, Resolution.Daily);

// QuoteBar objects
var historyQuoteBars = History<QuoteBar>(_symbol, 100, Resolution.Minute);

// Tick objects
var historyTicks = History<Tick>(_symbol, TimeSpan.FromSeconds(10), Resolution.Tick);
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of Coinbase Crypto pairs, call the `AddUniverse` method with a `CryptoUniverse` object. A Crypto universe uses a selection function to select Crypto pairs based on their OHLCV and dollar volume of the previous day as of midnight Coordinated Universal Time (UTC).

```C#
using QuantConnect.Data.UniverseSelection;

public override void Initialize()
{
    UniverseSettings.Asynchronous = true;
    SetBrokerageModel(BrokerageName.Coinbase, AccountType.Cash);
    _universe = AddUniverse(CryptoUniverse.Coinbase(UniverseSelectionFilter));
}

private IEnumerable<Symbol> UniverseSelectionFilter(IEnumerable<CryptoUniverse> universeDay)
{
    return from c in universeDay
           where c.Volume >= 100m && c.VolumeInUsd > 10000m
           select c.Symbol;
}
```

For more information about universe settings, see Settings .

Universe History

You can get historical universe data in an algorithm and in the Research Environment.

Historical Universe Data in Algorithms

To get historical universe data in an algorithm, call the `History` method with the `Universe` object, and the lookback period. If there is no data in the period you request, the history result is empty.

```C#
var history = History(_universe, 30, Resolution.Daily);
foreach (var universeDay in history)
{
    foreach (CryptoUniverse universeItem in universeDay)
    {
        Log($"{universeItem.Symbol} price at {universeItem.EndTime}: {universeItem.Close}");
    }
}
```

Historical Universe Data in Research

To get historical universe data in research, call the `UniverseHistory` method with the `Universe` object, and the lookback period. The `UniverseHistory` returns the filtered universe. If there is no data in the period you request, the history result is empty.

```C#
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var universeDay in universeHistory)
{
    foreach (CryptoUniverse universeItem in universeDay)
    {
        Console.WriteLine($"{universeItem.Symbol} price at {universeItem.EndTime}: {universeItem.Close}");
    }
}
```

You can call the `History` method in Research.

Remove Subscriptions

To unsubscribe from a Crypto pair that you added with the `AddCrypto` method, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_symbol);
```

The `RemoveSecurity` method cancels your open orders for the security and liquidates your holdings in the virtual pair .

Supported Assets

The following table shows the available Cryptocurrency pairs:

Pairs Available (970)					
00USD	00USDC	1INCHBTC	1INCHEUR	1INCHGBP	1INCHUSD
1INCHUSDC	A8USD	A8USDC	AAVEBTC	AAVEEUR	AAVEGBP
AAVEUSD	AAVEUSDC	ABTUSD	ABTUSDC	ACHUSD	ACHUSDC
ACSUSD	ACSUSDC	ACXUSD	ACXUSDC	ADABTC	ADAETH
ADAEUR	ADAGBP	ADAUSD	ADAUSDC	ADAUSDT	AERGOUSD
AERGOUSDC	AEROUSD	AEROUSDC	AGLDUSD	AGLDUSDC	AGLDUSDT
AIOZUSD	AIOZUSDC	AKTUSD	AKTUSDC	ALCXEUR	ALCXUSD
ALCXUSDC	ALCXUSDT	ALEOUSD	ALEOUSDC	ALEPHUSD	ALEPHUSDC
ALGOBTC	ALGOEUR	ALGOGBP	ALGOUSD	ALGOUSDC	ALICEUSD
ALICEUSDC	ALTUSD	ALTUSDC	AMPUSD	AMPUSDC	ANKRBTC
ANKREUR	ANKRGBP	ANKRUSD	ANKRUSDC	ANTUSD	ANTUSDC
APEEUR	APEUSD	APEUSDC	APEUSDT	API3USD	API3USDC
API3USDT	APTUSD	APTUSDC	APTUSDT	ARBUSD	ARBUSDC
ARKMUSD	ARKMUSDC	ARPAEUR	ARPAUSD	ARPAUSDC	ARPAUSDT
ASMUSD	ASMUSDC	ASMUSDT	ASTUSD	ASTUSDC	ATAUSD
ATAUSDC	ATHUSD	ATHUSDC	ATOMBTC	ATOMEUR	ATOMGBP
ATOMUSD	ATOMUSDC	ATOMUSDT	AUCTIONEUR	AUCTIONUSD	AUCTIONUSDC
AUCTIONUSDT	AUDIOUSD	AUDIOUSDC	AURORAUSD	AURORAUSDC	AVAXBTC
AVAXEUR	AVAXUSD	AVAXUSDC	AVAXUSDT	AVTUSD	AVTUSDC
AXLUSD	AXLUSDC	AXSBTC	AXSEUR	AXSUSD	AXSUSDC
AXSUSDT	B3USD	B3USDC	BADGEREUR	BADGERUSD	BADGERUSDC
BADGERUSDT	BALBTC	BALUSD	BALUSDC	BANDBTC	BANDEUR
BANDGBP	BANDUSD	BANDUSDC	BATBTC	BATETH	BATEUR
BATUSD	BATUSDC	BCHBTC	BCHEUR	BCHGBP	BCHUSD

BCHUSDC	BERAUSD	BERAUSDC	BICOEUR	BICOUSD	BICOUSDC
BICOUSDT	BIGTIMEUSD	BIGTIMEUSDC	BITUSD	BITUSDC	BLASTUSD
BLASTUSDC	BLURUSD	BLURUSDC	BLZUSD	BLZUSDC	BNTBTC
BNTEUR	BNTGBP	BNTUSD	BNTUSDC	BOBAUSD	BOBAUSDC
BONDUSD	BONKUSD	BONKUSDC	BTCEUR	BTCGBP	BTCUSD
BTCUSDC	BTCUSDT	BTCUST	BTRSTBTC	BTRSTEUR	BTRSTGBP
BTRSTUSD	BTRSTUSDC	BTRSTUSDT	C98USD	C98USDC	CBETHETH
CBETHUSD	CBETHUSDC	CELRUSD	CELRUSDC	CGLDBTC	CGLDEUR
CGLDGBP	CGLDUSD	CGLDUSDC	CHZEUR	CHZGBP	CHZUSD
CHZUSDC	CHZUSDT	CLANKERUSD	CLANKERUSDC	CLVEUR	CLVGBP
CLVUSD	CLVUSDC	CLVUSDT	COMPBTC	COMPUSD	COMPUSDC
COOKIEUSD	COOKIEUSDC	CORECHAINUSD	CORECHAINUSDC	COTIUSD	COTIUSDC
COVALUSD	COVALUSDC	COVALUSDT	COWUSD	COWUSDC	CROEUR
CROUSD	CROUSDC	CROUSDT	CRVBTC	CRVEUR	CRVGBP
CRVUSD	CRVUSDC	CTSIBTC	CTSIUSD	CTSIUSDC	CTXEUR
CTXUSD	CTXUSDC	CTXUSDT	CVCUSD	CVCUSDC	CVXUSD
CVXUSDC	DAIUSD	DAIUSDC	DARUSD	DARUSDC	DASHBTC
DASHUSD	DASHUSDC	DDXEUR	DDXUSD	DDXUSDT	DEGENUSD
DEGENUSDC	DESOEUR	DESOUSD	DESOUSDC	DESOUSDT	DEXTUSD
DEXTUSDC	DIAEUR	DIAUSD	DIAUSDC	DIAUSDT	DIMOUSD
DIMOUSDC	DNTUSD	DNTUSDC	DOGEBTC	DOGEEUR	DOGEGBP
DOGEUSD	DOGEUSDC	DOGEUSDT	DOGINMEUSD	DOGINMEUSDC	DOTBTC
DOTEUR	DOTGBP	DOTUSD	DOTUSDC	DOTUSDT	DREPUSD
DREPUSDC	DRIFTUSD	DRIFTUSDC	DYPUSD	DYPUSDC	EDGEUSD
EDGEUSDC	EGLDUSD	EGLDUSDC	EIGENUSD	EIGENUSDC	ELAUSD
ELAUSDC	ENJBTC	ENJUSD	ENJUSDC	ENJUSDT	ENSEUR
ENSUSD	ENSUSDC	ENSUSDT	EOSBTC	EOSEUR	EOSUSD
EOSUSDC	ERNUSD	ERNUSDC	ETCBTC	ETCEUR	ETCGBP

ETCUSD	ETCUSDC	ETHBTC	ETHDAI	ETHEUR	ETHFIUSD
ETHFIUSDC	ETHGBP	ETHUSD	ETHUSDC	ETHUSDT	EURCEUR
EURCUSD	EURCUSDC	EUROCEUR	EUROCUSD	EUROCUSDC	FAIUSD
FAIUSDC	FARMUSD	FARMUSDC	FARMUSDT	FETUSD	FETUSDC
FETUSDT	FIDAEUR	FIDAUSD	FIDAUSDC	FIDAUSDT	FILBTC
FILEUR	FILGBP	FILUSD	FILUSDC	FISUSD	FISUSDC
FLOKIUSD	FLOKIUSDC	FLOWUSD	FLOWUSDC	FLOWUSDT	FLRUSD
FLRUSDC	FORTHBTC	FORTHEUR	FORTHGBP	FORTHUSD	FORTHUSDC
FORTUSD	FORTUSDC	FOXUSD	FOXUSDC	FOXUSDT	FXUSD
FXUSDC	GALAEUR	GALAUSD	GALAUSDC	GALAUSDT	GALUSD
GALUSDC	GFIUSD	GFIUSDC	GHSTUSD	GHSTUSDC	GIGAUSD
GIGAUSDC	GLMUSD	GLMUSDC	GMTUSD	GMTUSDC	GMTUSDT
GNOUSD	GNOUSDC	GNTUSDC	GODSUSD	GODSUSDC	GRTBTC
GRTEUR	GRTGBP	GRTUSD	GRTUSDC	GSTUSD	GSTUSDC
GTCUSD	GTCUSDC	GUSD	GUSDC	GUSDUSD	GUSDUSDC
GYENUSD	GYENUSDC	HBARUSD	HBARUSDC	HBARUSDT	HFTUSD
HFTUSDC	HIGHUSD	HIGHUSDC	HNTUSD	HNTUSDC	HONEYUSD
HONEYUSDC	HOPRUSD	HOPRUSDC	ICPBTC	ICPEUR	ICPGBP
ICPUSD	ICPUSDC	ICPUSDT	IDEXUSD	IDEXUSDC	IDEXUSDT
ILVUSD	ILVUSDC	IMXUSD	IMXUSDC	IMXUSDT	INDEXUSD
INDEXUSDC	INJUSD	INJUSDC	INVUSD	INVUSDC	IOTXEUR
IOTXUSD	IOTXUSDC	IOUSD	IOUSDC	IPUSD	IPUSDC
JASMYUSD	JASMYUSDC	JASMYUSDT	JTOUSD	JTOUSDC	KAITOUSD
KAITOUSDC	KARRATUSD	KARRATUSDC	KAVAUSD	KAVAUSDC	KEEPUSD
KERNELUSD	KERNELUSDC	KEYCATUSD	KEYCATUSDC	KNCBTC	KNCUSD
KNCUSDC	KRLEUR	KRLUSD	KRLUSDC	KRLUSDT	KSMUSD
KSMUSDC	L3USD	L3USDC	LCXEUR	LCXUSD	LCXUSDC
LCXUSDT	LDOUSD	LDOUSDC	LINKBTC	LINKETH	LINKEUR

LINKGBP	LINKUSD	LINKUSDC	LINKUSDT	LITUSD	LITUSDC
LOKAUSD	LOKAUSDC	LOOMUSD	LOOMUSDC	LPTUSD	LPTUSDC
LQTYEUR	LQTYUSD	LQTYUSDC	LQTYUSDT	LRCBTC	LRCUSD
LRCUSDC	LRCUSDT	LRDSUSD	LRDSUSDC	LSETHETH	LSETHUSD
LSETHUSDC	LTCBTC	LTCEUR	LTCGBP	LTCUSD	LTCUSDC
MAGICUSD	MAGICUSDC	MANABTC	MANAETH	MANAEUR	MANAUSD
MANAUSDC	MANTLEUSD	MANTLEUSDC	MASKEUR	MASKGBP	MASKUSD
MASKUSDC	MASKUSDT	MATHUSD	MATHUSDC	MATICBTC	MATICEUR
MATICGBP	MATICUSD	MATICUSDC	MATICUSDT	MCO2USD	MCO2USDT
MDTUSD	MDTUSDC	MDTUSDT	MEDIAUSD	MEDIAUSDC	METISUSD
METISUSDC	MEUSD	MEUSDC	MINAEUR	MINAUSD	MINAUSDC
MINAUSDT	MIRBTC	MIREUR	MIRGBP	MIRUSD	MKRBTC
MKRUSD	MKRUSDC	MLNUSD	MLNUSDC	MNDEUSD	MNDEUSDC
MOBILEUSD	MOBILEUSDC	MOGUSD	MOGUSDC	MONAUSD	MONAUSDC
MOODENGUSD	MOODENGUSDC	MORPHOUSD	MORPHOUSDC	MOVEUSD	MOVEUSDC
MPLUSD	MPLUSDC	MSOLUSD	MSOLUSDC	MTLUSD	MTLUSDC
MUSDUSD	MUSEUSD	MUSEUSDC	NCTEUR	NCTUSD	NCTUSDC
NCTUSDT	NEARUSD	NEARUSDC	NEARUSDT	NEONUSD	NEONUSDC
NKNBTC	NKNEUR	NKNGBP	NKNUSD	NKNUSDC	NMRBTC
NMREUR	NMRGBP	NMRUSD	NMRUSDC	NUBTC	NUEUR
NUGBP	NUUSD	OCEANUSD	OCEANUSDC	OGNBTC	OGNUSD
OGNUSDC	OMGBTC	OMGEUR	OMGGBP	OMGUSD	OMNIUSD
OMNIUSDC	ONDOUSD	ONDOUSDC	OPUSD	OPUSDC	OPUSDT
ORCAUSD	ORCAUSDC	ORNBTC	ORNUSD	ORNUSDC	ORNUSDT
OSMOUSD	OSMOUSDC	OXTUSD	OXTUSDC	PAXGUSD	PAXGUSDC
PAXUSD	PAXUSDC	PAXUSDT	PENDLEUSD	PENDLEUSDC	PENGUUSD
PENGUUSDC	PEPEUSD	PEPEUSDC	PERPEUR	PERPUSD	PERPUSDC
PERPUSDT	PIRATEUSD	PIRATEUSDC	PLAUSD	PLAUSDC	PLUUSD

PLUUSDC	PNGUSD	PNGUSDC	PNUTUSD	PNUTUSDC	POLSUSD
POLSUSDC	POLSUSDT	POLUSD	POLUSDC	POLYUSD	POLYUSDT
PONDUSD	PONDUSDC	POPCATUSD	POPCATUSDC	POWREUR	POWRUSD
POWRUSDC	POWRUSDT	PRCLUSD	PRCLUSDC	PRIMEUSD	PRIMEUSDC
PROMPTUSD	PROMPTUSDC	PROUSD	PROUSDC	PRQUSD	PRQUSDC
PUNDIXUSD	PUNDIXUSDC	PYRUSD	PYRUSDC	PYTHUSD	PYTHUSDC
PYUSDUSD	PYUSDUSDC	QIUSD	QIUSDC	QNTUSD	QNTUSDC
QNTUSDT	QSPUSD	QSPUSDT	QUICKUSD	RADBTC	RADEUR
RADGBP	RADUSD	RADUSDC	RADUSDT	RAIUSD	RAIUSDC
RAREUSD	RAREUSDC	RARIUSD	RARIUSDC	RBNUSD	RBNUSDC
REDUSD	REDUSDC	RENBTC	RENDERUSD	RENDERUSDC	RENUSD
RENUSDC	REPBTC	REPUSD	REQBTC	REQEUR	REQGBP
REQUSD	REQUSDC	REQUSDT	REZUSD	REZUSDC	RGTUSD
RLCBTC	RLCUSD	RLCUSDC	RLYEUR	RLYGBP	RLYUSD
RLYUSDT	RNDREUR	RNDRUSD	RNDRUSDC	RNDRUSDT	RONINUSD
RONINUSDC	ROSEUSD	ROSEUSDC	ROSEUSDT	RPLUSD	RPLUSDC
RSRUSD	RSRUSDC	SAFEUSD	SAFEUSDC	SANDUSD	SANDUSDC
SANDUSDT	SDUSD	SDUSDC	SEAMUSD	SEAMUSDC	SEIUSD
SEIUSDC	SHDWUSD	SHDWUSDC	SHIBEUR	SHIBGBP	SHIBUSD
SHIBUSDC	SHIBUSDT	SHPINGEUR	SHPINGUSD	SHPINGUSDC	SHPINGUSDT
SKLBTC	SKLEUR	SKLGBP	SKLUSD	SKLUSDC	SNTUSD
SNTUSDC	SNXBTC	SNXEUR	SNXGBP	SNXUSD	SNXUSDC
SOLBTC	SOLETH	SOLEUR	SOLGBP	SOLUSD	SOLUSDC
SOLUSDT	SPAUSD	SPAUSDC	SPELLUSD	SPELLUSDC	SPELLUSDT
STGUSD	STGUSDC	STORJBTC	STORJUSD	STORJUSDC	STRKUSD
STRKUSDC	STXUSD	STXUSDC	STXUSDT	SUIUSD	SUIUSDC
SUKUEUR	SUKUUSD	SUKUUSDC	SUKUUSDT	SUPERUSD	SUPERUSDC
SUPERUSDT	SUSHIBTC	SUSHIETH	SUSHIEUR	SUSHIGBP	SUSHIUSD

SUSHIUSDC	SWELLUSD	SWELLUSDC	SWFTCUSD	SWFTCUSDC	SXTUSD
SXTUSDC	SYLOUSD	SYLOUSDC	SYNUSD	SYNUSDC	SYRUPUSD
SYRUPUSDC	TAOUSD	TAOUSDC	TIAUSD	TIAUSDC	TIMEUSD
TIMEUSDC	TNSRUSD	TNSRUSDC	TOSHIUSD	TOSHIUSDC	TRACEUR
TRACUSD	TRACUSDC	TRACUSDT	TRBBTC	TRBUSD	TRBUSDC
TRIBEUSD	TRUBTC	TRUEUR	TRUMPUSD	TRUMPUSDC	TRUUSD
TRUUSDC	TRUUSDT	TURBOUSD	TURBOUSDC	TUSD	TUSDC
TVKUSD	TVKUSDC	UMABTC	UMAEUR	UMAGBP	UMAUSD
UMAUSDC	UNFIUSD	UNFIUSDC	UNIBTC	UNIEUR	UNIGBP
UNIUSD	UNIUSDC	UPIUSD	UPIUSDT	USDCEUR	USDCGBP
USDTEUR	USDTGBP	USDTUSD	USDTUSDC	USTEUR	USTUSD
USTUSDT	VARAUSD	VARAUSDC	VELOUSD	VELOUSDC	VETUSD
VETUSDC	VGXEUR	VGXUSD	VGXUSDT	VOXELUSD	VOXELUSDC
VTHOUSD	VTHOUSDC	VVVUSD	VVVUSDC	WAMPLUSD	WAMPLUSDC
WAXLUSD	WAXLUSDC	WBTCBTC	WBTCUSD	WBTCUSDC	WCFGBTC
WCFGEUR	WCFGGBP	WCFGUSD	WCFGUSDC	WCFGUSDT	WELLUSD
WELLUSDC	WIFUSD	WIFUSDC	WLDUSD	WLDUSDC	WLUNABTC
WLUNAEUR	WLUNAGBP	WLUNAUSD	WLUNAUSDT	XCNUSD	XCNUSDC
XLMBTC	XLMEUR	XLMUSD	XLMUSDC	XLMUSDT	XRPBTC
XRPEUR	XRPGBP	XRPUSD	XRPUSDC	XRPUSDT	XTZBTC
XTZEUR	XTZGBP	XTZUSD	XTZUSDC	XYOBTC	XYOEUR
XYOUSD	XYOUSDC	XYOUSDT	YFIBTC	YFIIUSD	YFIUSD
YFIUSDC	ZECBTC	ZECUSD	ZECUSDC	ZENBTC	ZENUSD
ZENUSDC	ZENUSDT	ZETACHAINUSD	ZETACHAINUSDC	ZETAUSD	ZETAUSDC
ZKUSD	ZKUSDC	ZORAUSD	ZORAUSDC	ZROUSD	ZROUSDC
ZRXBTC	ZRXEUR	ZRXUSD	ZRXUSDC		

Example Applications

The Coinbase Crypto Price dataset enables you to accurately design strategies for Cryptocurrencies. Examples include the following strategies:

- Buy and hold
- Trading Cryptocurrency volatility and price action
- Allocating a small portion of your portfolio to Cryptocurrencies to hedge against inflation

Classic Algorithm Example

The following example algorithm buys and holds Bitcoin through the Coinbase exchange:

C#

```
public class CoinAPIDataAlgorithm : QCAlgorithm
{
    private Crypto _btc;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        UniverseSettings.Asynchronous = true;
        SetCash(100000);
        // Coinbase accepts Cash account type only. AccountType.Margin
        // results in an exception.
        SetBrokerageModel(BrokerageName.Coinbase, AccountType.Cash);
        // Add BTCUSD data.
        _btc = AddCrypto("BTCUSD", Resolution.Minute, Market.Coinbase);
        // Historical data
        var history = History(_btc.Symbol, 30, Resolution.Daily);
        // Add a Crypto universe that selects trading pairs on Coinbase.
        var universe = AddUniverse(CryptoUniverse.Coinbase(SelectAssets));
        // Historical Universe data
        var universeHistory = History(universe, 30, Resolution.Daily);
        foreach (var universeDay in universeHistory)
        {
            foreach (CryptoUniverse universeItem in universeDay)
            {
                var symbol = universeItem.Symbol;
                var price = universeItem.Close;
            }
        }
    }

    private IEnumerable<Symbol> SelectAssets(IEnumerable<CryptoUniverse> universeDay)
    {
        // Filter for materially traded Crypto pairs with significant
        // size and dollar volume, assuming higher capital flow in for
        // higher return.
        return from universeItem in universeDay
                where universeItem.Volume >= 100m
                && universeItem.VolumeInUsd > 10000000m
                select universeItem.Symbol;
    }

    public override void OnData(Slice slice)
    {
        // Speculate-invest all available free cash on BTCUSD, repecting
        // the order quantity restrictions to avoid invalid orders.
        if (Portfolio.CashBook["BTC"].Amount != 0)
        {
            return;
        }
        // Define a buffer so that we can scale down the order size,
        // avoiding trading errors.
        var buffer = Math.Max(CoinbaseFeeModel.TakerAdvanced1, Settings.FreePortfolioValuePercentage);
        var quantity = Portfolio.CashBook["USD"].Amount / _btc.AskPrice * (1-buffer);
        // Check if the quantity exceeds the minimum order size. On
        // Coinbase, the minimum order size is measured in the base
        // currency (BTC in this case).
        if (Math.Abs(quantity) < _btc.SymbolProperties.MinimumOrderSize)
        {
            return;
        }
        MarketOrder(_btc, quantity);
    }
}
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of Crypto pairs on the Coinbase exchange and then forms a equal-weighted portfolio of all the pairs in the universe:

```C#
public class CoinAPIDataAlgorithm : QCAlgorithm
{
    private List<string> _marketPairs;

    public override void Initialize()
    {
        SetStartDate(2020, 6, 1);
        SetEndDate(2021, 2, 1);
        SetCash(100000);
        Settings.FreePortfolioValuePercentage = 0.05m;
        // Warm up the security with the last known price to avoid conversion error
        Settings.SeedInitialPrices = true;
        UniverseSettings.Asynchronous = true;
        UniverseSettings.Resolution = Resolution.Minute;
        // Add Crypto Universe Selection that select crypto pairs in Coinbase exchange
        AddUniverse(CryptoUniverse.Coinbase(SelectAssets));
        // Get the pairs on Coinbase that have USD as the quote currency.
        _marketPairs = SymbolPropertiesDatabase.GetSymbolPropertiesList(Market.Coinbase)
            .Where(x => x.Value.QuoteCurrency == AccountCurrency)
            .Select(x => x.Key.Symbol)
            .ToList();
        AddAlpha( new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromHours(4), 0.025, null)
);
        // Equally invest to evenly dissipate the capital concentration risk of inidividual crypto pair
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }

    private IEnumerable<Symbol> SelectAssets(IEnumerable<CryptoUniverse> universeDay)
    {
        // Select pairs where the quote currency is USD.
        return universeDay.Where(c => _marketPairs.Contains(c.Symbol.Value))
            // Filter for materially traded crypto pairs with significant size and dollar volume, assuming higher
capital flow in for higher return
            .OrderBy(c => c.VolumeInUsd)
            .TakeLast(3)
            .Select(c => c.Symbol);
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach(var security in changes.AddedSecurities)
        {
            // Historical data
            var history = History(security.Symbol, 30, Resolution.Daily);
            Debug($"We got {history.Count()} items from our history request");
        }
    }
}
```

Research Example

The following example lists crypto-currency pairs with the greatest dollar volume in the Coinbase exchange:

```C#
var qb = new QuantBook();

// Add Cryptocurrency pair
var symbol = qb.AddCrypto("BTCUSDT", market:Market.Coinbase).Symbol;

// Historical data
var history = qb.History(symbol, 30, Resolution.Daily);
foreach (var bar in history)
{
    Console.WriteLine($"{bar.EndTime} {bar}");
}

// Add Crypto Universe Selection
IEnumerable<Symbol> UniverseSelectionFilter(IEnumerable<CryptoUniverse> universeDay)
{
    return universeDay
        .Where(x => x.VolumeInUsd != null && x.VolumeInUsd > 10000m)
        .OrderByDescending(x => x.VolumeInUsd)
        .Take(5)
        .Select(x => x.Symbol);
}
var universe = qb.AddUniverse(CryptoUniverse.Coinbase(UniverseSelectionFilter));

// Historical Universe data
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var universeDay in universeHistory)
{
    Console.WriteLine($"=== {universeDay.First().EndTime} ===");
    foreach (CryptoUniverse universeItem in universeDay.OrderByDescending(x => x.VolumeInUsd))
    {
        Console.WriteLine($"{universeItem.Symbol}: {universeItem.VolumeInUsd}");
    }
}
```

Data Point Attributes

The Coinbase Crypto Price dataset provides `TradeBar` , `QuoteBar` , `Tick` , and `CryptoUniverse` objects.

TradeBar Attributes

`TradeBar` objects have the following attributes:

QuoteBar Attributes

`QuoteBar` objects have the following attributes:

Tick Attributes

`Tick` objects have the following attributes:

CryptoUniverse Attributes

`CryptoUniverse` objects have the following attributes:

CoinAPI

Kraken Crypto Price Data

Introduction

The Kraken Crypto Price Data by CoinAPI provides Cryptocurrency price and volume data points. The data covers 1,262 Cryptocurrency pairs, starts in October 2013, and is delivered on any frequency from tick to daily. This dataset is created by monitoring the trading activity on the Cryptocurrency markets/exchanges supported by QuantConnect.

For more information about the Kraken Crypto Price Data dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

CoinAPI was founded by Artur Pietrzyk in 2016 with the goal of providing real-time and historical cryptocurrency market data, collected from hundreds of exchanges. CoinAPI provides access to Cryptocurrencies for traders, market makers, and developers building third-party applications.

Getting Started

The following snippet demonstrates how to request data from the Kraken Crypto Price dataset:

```C#
// Kraken accepts both Cash and Margin type account.
SetBrokerageModel(BrokerageName.Kraken, AccountType.Cash);
//SetBrokerageModel(BrokerageName.Kraken, AccountType.Margin);

_symbol = AddCrypto("BTCUSD", Resolution.Minute, Market.Kraken).Symbol;

_universe = AddUniverse(CryptoUniverse.Kraken(UniverseSelectionFilter));
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	October 2013
Asset Coverage	1,262 Currency Pairs
Data Density	Dense
Resolution	Tick, Second, Minute, Hourly, & Daily
Timezone	UTC
Market Hours	Always Open

Requesting Data

To add Kraken Crypto Price data to your algorithm, call the `AddCrypto` method. Save a reference to the Crypto `Symbol` so you can access the data later in your algorithm.

```
C#
public class CoinAPIDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2020, 6, 1);
        SetEndDate(2021, 6, 1);
        SetCash(100000);

        // Kraken accepts both Cash and Margin type account.
        SetBrokerageModel(BrokerageName.Kraken, AccountType.Margin);

        _symbol = AddCrypto("BTCUSD", Resolution.Minute, Market.Kraken).Symbol;
    }
}
```

For more information about creating Crypto subscriptions, see Requesting Data .

Accessing Data

To get the current Kraken Crypto Price data, index the `Bars` , `QuoteBars` , or `Ticks` properties of the current `Slice` with the Crypto `Symbol` . `Slice` objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.Bars.ContainsKey(_symbol))
    {
        var tradeBar = slice.Bars[_symbol];
        Log($"{_symbol} price at {slice.Time}: {tradeBar.Close}");
    }

    if (slice.QuoteBars.ContainsKey(_symbol))
    {
        var quoteBar = slice.QuoteBars[_symbol];
        Log($"{_symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    if (slice.Ticks.ContainsKey(_symbol))
    {
        var ticks = slice.Ticks[_symbol];
        foreach (var tick in ticks)
        {
            Log($"{_symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

You can also iterate through all of the data objects in the current `Slice` .

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Bars)
    {
        var symbol = kvp.Key;
        var tradeBar = kvp.Value;
        Log($"{symbol} price at {slice.Time}: {tradeBar.Close}");
    }

    foreach (var kvp in slice.QuoteBars)
    {
        var symbol = kvp.Key;
        var quoteBar = kvp.Value;
        Log($"{symbol} bid at {slice.Time}: {quoteBar.Bid.Close}");
    }

    foreach (var kvp in slice.Ticks)
    {
        var symbol = kvp.Key;
        var ticks = kvp.Value;
        foreach (var tick in ticks)
        {
            Log($"{symbol} price at {slice.Time}: {tick.Price}");
        }
    }
}
```

For more information about accessing Crypto data, see Handling Data .

Historical Data

To get historical Kraken Crypto Price data, call the `History` method with the Crypto `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
// TradeBar objects
var historyTradeBars = History(_symbol, 100, Resolution.Daily);

// QuoteBar objects
var historyQuoteBars = History<QuoteBar>(_symbol, 100, Resolution.Minute);

// Tick objects
var historyTicks = History<Tick>(_symbol, TimeSpan.FromSeconds(10), Resolution.Tick);
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of Kraken Crypto pairs, call the `AddUniverse` method with a `CryptoUniverse` object. A Crypto universe uses a selection function to select Crypto pairs based on their OHLCV and dollar volume of the previous day as of midnight Coordinated Universal Time (UTC).

```C#
using QuantConnect.Data.UniverseSelection;

public override void Initialize()
{
    SetBrokerageModel(BrokerageName.Kraken, AccountType.Margin);
    UniverseSettings.Asynchronous = true;
    _universe = AddUniverse(CryptoUniverse.Kraken(UniverseSelectionFilter));
}

private IEnumerable<Symbol> UniverseSelectionFilter(IEnumerable<CryptoUniverse> universeDay)
{
    return from c in universeDay
           where c.Volume >= 100m && c.VolumeInUsd > 10000m
           select c.Symbol;
}
```

For more information about universe settings, see Settings .

Universe History

You can get historical universe data in an algorithm and in the Research Environment.

Historical Universe Data in Algorithms

To get historical universe data in an algorithm, call the `History` method with the `Universe` object, and the lookback period. If there is no data in the period you request, the history result is empty.

```C#
var history = History(_universe, 30, Resolution.Daily);
foreach (var universeDay in history)
{
    foreach (CryptoUniverse universeItem in universeDay)
    {
        Log($"{universeItem.Symbol} price at {universeItem.EndTime}: {universeItem.Close}");
    }
}
```

Historical Universe Data in Research

To get historical universe data in research, call the `UniverseHistory` method with the `Universe` object, and the lookback period. The `UniverseHistory` returns the filtered universe. If there is no data in the period you request, the history result is empty.

```C#
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var universeDay in universeHistory)
{
    foreach (CryptoUniverse universeItem in universeDay)
    {
        Console.WriteLine($"{universeItem.Symbol} price at {universeItem.EndTime}: {universeItem.Close}");
    }
}
```

You can call the `History` method in Research.

Remove Subscriptions

To unsubscribe from a Crypto pair that you added with the `AddCrypto` method, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_symbol);
```

The `RemoveSecurity` method cancels your open orders for the security and liquidates your holdings in the virtual pair .

Supported Assets

The following table shows the available Cryptocurrency pairs:

Pairs Available (1262)					
1INCHEUR	1INCHUSD	AAVEAUD	AAVEBTC	AAVEETH	AAVEEUR
AAVEGBP	AAVEUSD	ACAEUR	ACAUSD	ACHEUR	ACHUSD
ACTEUR	ACTUSD	ACXEUR	ACXUSD	ADAAUD	ADABTC
ADAETH	ADAEUR	ADAGBP	ADAUSD	ADAUSDC	ADAUSDT
ADXEUR	ADXUSD	AEROEUR	AEROUSD	AEUR	AEVOEUR
AEVOUSD	AGLDEUR	AGLDUSD	AI16ZEUR	AI16ZUSD	AI16ZUSDC
AI16ZUSDT	AIOZEUR	AIOZUSD	AIREUR	AIRUSD	AIXBTEUR
AIXBTUSD	AKTEUR	AKTUSD	ALCHEUR	ALCHUSD	ALCXEUR
ALCXUSD	ALGOBTC	ALGOETH	ALGOEUR	ALGOGBP	ALGOUSD
ALGOUSDC	ALGOUSDT	ALICEEUR	ALICEUSD	ALPHAEUR	ALPHAUSD
ALTEUR	ALTUSD	ANKRBTC	ANKREUR	ANKRGBP	ANKRUSD
ANLOGEUR	ANLOGUSD	ANONEUR	ANONUSD	ANTBTC	ANTETH
ANTEUR	ANTUSD	APEEUR	APENFTEUR	APENFTUSD	APEUSD
APEUSDC	APEUSDT	API3EUR	API3USD	APTEUR	APTUSD
APUEUR	APUUSD	ARBEUR	ARBUSD	ARCEUR	ARCUSD
AREUR	ARKMEUR	ARKMUSD	ARPAEUR	ARPAUSD	ARUSD
ASRREUR	ASRRUSD	ASTREUR	ASTRUSD	ATHEUR	ATHUSD
ATLASEUR	ATLASUSD	ATOMAUD	ATOMBTC	ATOMETH	ATOMEUR
ATOMGBP	ATOMUSD	ATOMUSDC	ATOMUSDT	AUCTIONEUR	AUCTIONUSD
AUDIOEUR	AUDIOUSD	AUDJPY	AUDUSD	AURAEUR	AURAUSD
AUSD	AVAAIEUR	AVAAIUSD	AVAXEUR	AVAXUSD	AVAXUSDC
AVAXUSDT	AXSEUR	AXSUSD	B3EUR	B3USD	BABYEUR
BABYUSD	BADGEREUR	BADGERUSD	BALBTC	BALETH	BALEUR
BALUSD	BANANAS31EUR	BANANAS31USD	BANDEUR	BANDUSD	BATBTC

BATETH	BATEUR	BATJPY	BATUSD	BCHAUD	BCHBTC
BCHETH	BCHEUR	BCHGBP	BCHJPY	BCHUSD	BCHUSDC
BCHUSDT	BDXNEUR	BDXNUSD	BEAMEUR	BEAMUSD	BERAEUR
BERAUSD	BERAUSDC	BERAUSDT	BICOEUR	BICOUSD	BIGTIMEEUR
BIGTIMEUSD	BIOEUR	BIOUSD	BITEUR	BITUSD	BLUREUR
BLURUSD	BLZEUR	BLZUSD	BMTEUR	BMTUSD	BNBEUR
BNBUSD	BNBUSDC	BNBUSDT	BNCEUR	BNCUSD	BNTBTC
BNTEUR	BNTGBP	BNTUSD	BOBAEUR	BOBAUSD	BODENEUR
BODENUSD	BONDEUR	BONDUSD	BONKEUR	BONKUSD	BRICKEUR
BRICKUSD	BSXEUR	BSXUSD	BTCAED	BTCAUD	BTCCAD
BTCCHF	BTCDAI	BTCEUR	BTCGBP	BTCJPY	BTCPYUSD
BTCUSD	BTCUSDC	BTCUSDQ	BTCUSDR	BTCUSDT	BTTEUR
BTTUSD	C98EUR	C98USD	CAKEEUR	CAKEUSD	CATEUR
CATUSD	CCDEUR	CCDUSD	CELOEUR	CELOUSD	CELREUR
CELRUSD	CFGEUR	CFGUSD	CHEEMSEUR	CHEEMSUSD	CHEXEUR
CHEXUSD	CHILLHOUSEEUR	CHILLHOUSEUSD	CHREUR	CHRUSD	CHZEUR
CHZUSD	CLANKEREUR	CLANKERUSD	CLOUDEUR	CLOUDUSD	CLVEUR
CLVUSD	CMETHETH	CMETHUSD	COMPBTC	COMPETH	COMPEUR
COMPUSD	COOKIEEUR	COOKIEUSD	COQEUR	COQUSD	CORNEUR
CORNUSD	COTIEUR	COTIUSD	COWEUR	COWUSD	CPOOLEUR
CPOOLUSD	CQTEUR	CQTUSD	CROEUR	CROUSD	CROUSDC
CROUSDT	CRVBTC	CRVETH	CRVEUR	CRVUSD	CSMEUR
CSMUSD	CTSIEUR	CTSIUSD	CVCEUR	CVCUSD	CVXEUR
CVXUSD	CXTEUR	CXTUSD	CYBEREUR	CYBERUSD	DAIEUR
DAIUSD	DAIUSDT	DASHBTC	DASHEUR	DASHUSD	DBREUR
DBRUSD	DEEPEUR	DEEPUSD	DEGENEUR	DEGENUSD	DENTEUR
DENTUSD	DMCEUR	DMCUSD	DOGEUR	DOGSEUR	DOGSUSD
DOGUSD	DOLOEUR	DOLOUSD	DOTAUD	DOTBTC	DOTETH

DOTEUR	DOTGBP	DOTJPY	DOTUSD	DOTUSDC	DOTUSDT
DRIFTEUR	DRIFTUSD	DRVEUR	DRVUSD	DUCKEUR	DUCKUSD
DYDXEUR	DYDXUSD	DYMEUR	DYMUSD	EDGEEUR	EDGEUSD
EGLDEUR	EGLDUSD	EIGENEUR	EIGENUSD	ELXEUR	ELXUSD
ENAEUR	ENAUSD	ENJBTC	ENJEUR	ENJGBP	ENJJPY
ENJUSD	ENSEUR	ENSUSD	EOSBTC	EOSETH	EOSEUR
EOSUSD	EOSUSDC	EOSUSDT	EPTEUR	EPTUSD	ESEUR
ESUSD	ESXEUR	ESXUSD	ETCBTC	ETCETH	ETCEUR
ETCUSD	ETH2.SETH	ETHAED	ETHAUD	ETHBTC	ETHCAD
ETHCHF	ETHDAI	ETHEUR	ETHFIEUR	ETHFIUSD	ETHGBP
ETHJPY	ETHPYUSD	ETHUSD	ETHUSDC	ETHUSDT	ETHWETH
ETHWEUR	ETHWUSD	EULEUR	EULUSD	EURAUD	EURCAD
EURCHF	EURGBP	EURJPY	EUROPEUR	EUROPUSD	EUROPUSDC
EURQEUR	EURQUSD	EURREUR	EURRUSD	EURRUSDC	EURRUSDT
EURTEUR	EURTUSD	EURTUSDT	EURUSD	EWTBTC	EWTEUR
EWTGBP	EWTUSD	FARMEUR	FARMUSD	FARTCOINEUR	FARTCOINUSD
FARTCOINUSDC	FARTCOINUSDT	FETEUR	FETUSD	FHEEUR	FHEUSD
FIDAEUR	FIDAUSD	FILAUD	FILBTC	FILETH	FILEUR
FILGBP	FILUSD	FISEUR	FISUSD	FLOKIEUR	FLOKIUSD
FLOWBTC	FLOWETH	FLOWEUR	FLOWGBP	FLOWUSD	FLREUR
FLRUSD	FLUXEUR	FLUXUSD	FLYEUR	FLYUSD	FORTHEUR
FORTHUSD	FTMEUR	FTMUSD	FWOGEUR	FWOGUSD	FXSEUR
FXSUSD	GAIAEUR	GAIAUSD	GALAEUR	GALAUSD	GALEUR
GALUSD	GARIEUR	GARIUSD	GBPUSD	GEUR	GFIEUR
GFIUSD	GHIBLIEUR	GHIBLIUSD	GHSTBTC	GHSTEUR	GHSTGBP
GHSTUSD	GIGAEUR	GIGAUSD	GLMREUR	GLMRUSD	GMTEUR
GMTUSD	GMXEUR	GMXUSD	GNOBTC	GNOETH	GNOEUR
GNOUSD	GOATEUR	GOATUSD	GOMININGEUR	GOMININGUSD	GRASSEUR

GRASSUSD	GRIFFAINEUR	GRIFFAINUSD	GRTAUD	GRTBTC	GRTETH
GRTEUR	GRTGBP	GRTUSD	GSTEUR	GSTUSD	GTCEUR
GTCUSD	GUNEUR	GUNUSD	GUSD	HBAREUR	HBARUSD
HDXEUR	HDXUSD	HFTEUR	HFTUSD	HIPPOEUR	HIPPOUSD
HMSTREUR	HMSTRUSD	HNTEUR	HNTUSD	HONEYEUR	HONEYUSD
HOUSEEUR	HOUSEUSD	HPOS10IEUR	HPOS10IUSD	ICNTEUR	ICNTUSD
ICPEUR	ICPUSD	ICXBTC	ICXETH	ICXEUR	ICXUSD
IDEXEUR	IDEXUSD	IMXEUR	IMXUSD	INITEUR	INITUSD
INJEUR	INJUSD	INTREUR	INTRUSD	IPEUR	IPUSD
JAILSTOOLEUR	JAILSTOOLUSD	JASMYEUR	JASMYUSD	JITOSOLEUR	JITOSOLSOL
JITOSOLUSD	JOEEUR	JOEUSD	JSTEUR	JSTUSD	JTOEUR
JTOUSD	JUNOEUR	JUNOUSD	JUPEUR	JUPUSD	KAITOEUR
KAITOUSD	KAREUR	KARUSD	KASEUR	KASUSD	KAVABTC
KAVAETH	KAVAEUR	KAVAUSD	KEEPBTC	KEEPETH	KEEPEUR
KEEPUSD	KERNELEUR	KERNELUSD	KETEUR	KETUSD	KEUR
KEYEUR	KEYUSD	KILTEUR	KILTUSD	KINEUR	KINTEUR
KINTUSD	KINUSD	KMNOEUR	KMNOUSD	KNCBTC	KNCETH
KNCEUR	KNCUSD	KOBANEUR	KOBANUSD	KP3REUR	KP3RUSD
KSMAUD	KSMBTC	KSMDOT	KSMETH	KSMEUR	KSMGBP
KSMUSD	KTAEUR	KTAUSD	KUJIEUR	KUJIUSD	KUSD
L3EUR	L3USD	LAYEREUR	LAYERUSD	LCXEUR	LCXUSD
LDOEUR	LDOUSD	LINKAUD	LINKBTC	LINKETH	LINKEUR
LINKGBP	LINKJPY	LINKUSD	LINKUSDC	LINKUSDT	LITEUR
LITUSD	LMWREUR	LMWRUSD	LOCKINEUR	LOCKINUSD	LOFIEUR
LOFIUSD	LPTBTC	LPTEUR	LPTGBP	LPTUSD	LQTYEUR
LQTYUSD	LRCEUR	LRCUSD	LSETHETH	LSETHEUR	LSETHUSD
LSKBTC	LSKETH	LSKEUR	LSKUSD	LSSOLEUR	LSSOLSOL
LSSOLUSD	LTCAUD	LTCBTC	LTCETH	LTCEUR	LTCGBP

LTCJPY	LTCUSD	LTCUSDC	LTCUSDT	LUNA2EUR	LUNA2USD
LUNAEUR	LUNAUSD	MANABTC	MANAETH	MANAEUR	MANAUSD
MANAUSDC	MANAUSDT	MASKEUR	MASKUSD	MATEUR	MATICBTC
MATICEUR	MATICGBP	MATICPOL	MATICUSD	MATICUSDC	MATICUSDT
MATUSD	MCEUR	MCUSD	MEEUR	MELANIAEUR	MELANIAUSD
MELANIAUSDC	MELANIAUSDT	MEMEEUR	MEMEUSD	MERLEUR	MERLUSD
METHETH	METHUSD	METISEUR	METISUSD	MEUR	MEUSD
MEWEUR	MEWUSD	MICHIEUR	MICHIUSD	MINABTC	MINAEUR
MINAGBP	MINAUSD	MIRAEUR	MIRAUSD	MIREUR	MIRUSD
MKRBTC	MKREUR	MKRGBP	MKRUSD	MLNBTC	MLNETH
MLNEUR	MLNUSD	MNGOEUR	MNGOUSD	MNTEUR	MNTUSD
MOGEUR	MOGUSD	MOODENGEUR	MOODENGUSD	MOONEUR	MOONUSD
MORPHOEUR	MORPHOUSD	MOVEEUR	MOVEUSD	MOVREUR	MOVRUSD
MSOLEUR	MSOLUSD	MUBARAKEUR	MUBARAKUSD	MULTIEUR	MULTIUSD
MUSD	MVEUR	MVUSD	MXCEUR	MXCUSD	NANOBTC
NANOETH	NANOEUR	NANOUSD	NEAREUR	NEARUSD	NEIROEUR
NEIROUSD	NILEUR	NILUSD	NMREUR	NMRUSD	NODLEUR
NODLUSD	NOSEUR	NOSUSD	NOTEUR	NOTUSD	NPCEUR
NPCUSD	NTRNEUR	NTRNUSD	NYMBTC	NYMEUR	NYMUSD
OCEANBTC	OCEANEUR	OCEANGBP	OCEANUSD	ODOSEUR	ODOSUSD
OGNEUR	OGNUSD	OMEUR	OMGBTC	OMGETH	OMGEUR
OMGJPY	OMGUSD	OMNIEUR	OMNIUSD	OMUSD	ONDOEUR
ONDOUSD	OPEUR	OPUSD	ORCAEUR	ORCAUSD	ORDEREUR
ORDERUSD	OSMOEUR	OSMOUSD	OXTBTC	OXTETH	OXTEUR
OXTUSD	OXYEUR	OXYUSD	PARTIEUR	PARTIUSD	PAXGBTC
PAXGETH	PAXGEUR	PAXGUSD	PDAEUR	PDAUSD	PEAQEUR
PEAQUSD	PENDLEEUR	PENDLEUSD	PENGUEUR	PENGUUSD	PENGUUSDC
PENGUUSDT	PEPEAUD	PEPECAD	PEPEEUR	PEPEGBP	PEPEUSD

PERPEUR	PERPUSD	PHAEUR	PHAUSD	PLUMEEUR	PLUMEUSD
PNUTEUR	PNUTUSD	POLEUR	POLISEUR	POLISUSD	POLSEUR
POLSUSD	POLUSD	PONDEUR	PONDUSD	PONKEEUR	PONKEUSD
POPCATEUR	POPCATGBP	POPCATUSD	PORTALEUR	PORTALUSD	POWREUR
POWRUSD	PRCLEUR	PRCLUSD	PRIMEEUR	PRIMEUSD	PROEUR
PROMPTEUR	PROMPTUSD	PROUSD	PSTAKEEUR	PSTAKEUSD	PUFFEREUR
PUFFERUSD	PUMPEUR	PUMPSOL	PUMPUSD	PYTHEUR	PYTHUSD
PYUSDEUR	PYUSDUSD	QIEUR	QIUSD	QNTEUR	QNTUSD
QTUMBTC	QTUMETH	QTUMEUR	QTUMUSD	RADEUR	RADUSD
RAIINEUR	RAIINUSD	RAREEUR	RAREUSD	RARIBTC	RARIEUR
RARIGBP	RARIUSD	RAYEUR	RAYUSD	RBCEUR	RBCUSD
REDEUR	REDUSD	RENBTC	RENDEREUR	RENDERUSD	RENEUR
RENGBP	RENUSD	REPBTC	REPETH	REPEUR	REPUSD
REPV2BTC	REPV2ETH	REPV2EUR	REPV2USD	REQEUR	REQUSD
RETARDIOEUR	RETARDIOUSD	REZEUR	REZUSD	RIZEEUR	RIZEUSD
RLCEUR	RLCUSD	RLUSDEUR	RLUSDUSD	RLUSDUSDC	RLUSDUSDT
RNDREUR	RNDRUSD	ROOKEUR	ROOKUSD	RPLEUR	RPLUSD
RSREUR	RSRUSD	RUJIEUR	RUJIUSD	RUNEEUR	RUNEUSD
SAFEEUR	SAFEUSD	SAGAEUR	SAGAUSD	SAHARAEUR	SAHARAUSD
SAMOEUR	SAMOUSD	SANDBTC	SANDEUR	SANDGBP	SANDUSD
SBREUR	SBRUSD	SCBTC	SCETH	SCEUR	SCRTEUR
SCRTUSD	SCUSD	SDNEUR	SDNUSD	SEIEUR	SEIUSD
SEUR	SGBEUR	SGBUSD	SHIBEUR	SHIBUSD	SHIBUSDC
SHIBUSDT	SIGMAEUR	SIGMAUSD	SKYEUR	SKYUSD	SNEKEUR
SNEKUSD	SNXAUD	SNXBTC	SNXETH	SNXEUR	SNXGBP
SNXUSD	SOGNIEUR	SOGNIUSD	SOLAUD	SOLBTC	SOLCAD
SOLETH	SOLEUR	SOLGBP	SOLUSD	SOLUSDC	SOLUSDT
SONICEUR	SONICUSD	SOSOEUR	SOSOUSD	SPELLEUR	SPELLUSD

SPICEEUR	SPICEUSD	SPKEUR	SPKUSD	SPXEUR	SPXUSD
SRMBTC	SRMEUR	SRMGBP	SRMUSD	SSVEUR	SSVUSD
STEPEUR	STEPUSD	STGEUR	STGUSD	STORJBTC	STORJETH
STORJEUR	STORJUSD	STRDEUR	STRDUSD	STRKEUR	STRKUSD
STXEUR	STXUSD	SUIEUR	SUIGBP	SUIUSD	SUNDOGEUR
SUNDOGUSD	SUNEUR	SUNUSD	SUPEREUR	SUPERUSD	SUSD
SUSDC	SUSDT	SUSHIBTC	SUSHIEUR	SUSHIGBP	SUSHIUSD
SWARMSEUR	SWARMSUSD	SWEATEUR	SWEATUSD	SWELLEUR	SWELLUSD
SXTEUR	SXTUSD	SYNEUR	SYNUSD	SYRUPEUR	SYRUPUSD
TACEUR	TACUSD	TANSSIEUR	TANSSIUSD	TAOEUR	TAOUSD
TBTCBTC	TBTCETH	TBTCEUR	TBTCUSD	TEEREUR	TEERUSD
TERMEUR	TERMUSD	TEUR	TIAEUR	TIAUSD	TITCOINEUR
TITCOINUSD	TLMEUR	TLMUSD	TNSREUR	TNSRUSD	TOKEEUR
TOKENEUR	TOKENUSD	TOKEUSD	TONEUR	TONUSD	TONUSDC
TONUSDT	TOSHIEUR	TOSHIUSD	TRACEUR	TRACUSD	TREEEUR
TREEUSD	TREMPEUR	TREMPUSD	TRUEUR	TRUMPEUR	TRUMPUSD
TRUMPUSDC	TRUMPUSDT	TRUUSD	TRXBTC	TRXETH	TRXEUR
TRXUSD	TRXUSDD	TURBOEUR	TURBOUSD	TUSD	TUSDEUR
TUSDUSD	TVKEUR	TVKUSD	UFDEUR	UFDUSD	UMAEUR
UMAUSD	UNFIEUR	UNFIUSD	UNIBTC	UNIETH	UNIEUR
UNITEEUR	UNITEUSD	UNIUSD	USDAED	USDCAD	USDCAUD
USDCCAD	USDCCHF	USDCEUR	USDCGBP	USDCHF	USDCUSD
USDCUSDT	USDDEUR	USDDUSD	USDGEUR	USDGUSD	USDGUSDC
USDJPY	USDQEUR	USDQUSD	USDQUSDC	USDQUSDT	USDREUR
USDRUSD	USDRUSDC	USDRUSDT	USDSEUR	USDSUSD	USDTAUD
USDTCAD	USDTCHF	USDTEUR	USDTGBP	USDTJPY	USDTUSD
USDUCEUR	USDUCUSD	USTEUR	USTUSD	USTUSDC	USTUSDT
USUALEUR	USUALUSD	VANRYEUR	VANRYUSD	VELODROMEEUR	VELODROMEUSD

VINEEUR	VINEUSD	VIRTUALEUR	VIRTUALUSD	VIRTUALUSDC	VIRTUALUSDT
VSNEUR	VSNUSD	VVVEUR	VVVUSD	WALEUR	WALUSD
WAVESBTC	WAVESETH	WAVESEUR	WAVESUSD	WAXLEUR	WAXLUSD
WBTCBTC	WBTCEUR	WBTCUSD	WCTEUR	WCTUSD	WELLEUR
WELLUSD	WENEUR	WENUSD	WEUR	WIFEUR	WIFGBP
WIFUSD	WINEUR	WINUSD	WLDEUR	WLDUSD	WOOEUR
WOOUSD	WUSD	XCNEUR	XCNUSD	XDGAUD	XDGBTC
XDGCAD	XDGEUR	XDGGBP	XDGUSD	XDGUSDC	XDGUSDT
XLMAUD	XLMBTC	XLMEUR	XLMGBP	XLMUSD	XMRBTC
XMREUR	XMRUSD	XMRUSDC	XMRUSDT	XNYEUR	XNYUSD
XRPAUD	XRPBTC	XRPCAD	XRPETH	XRPEUR	XRPGBP
XRPJPY	XRPRLUSD	XRPUSD	XRPUSDC	XRPUSDT	XRTEUR
XRTUSD	XTZAUD	XTZBTC	XTZETH	XTZEUR	XTZGBP
XTZUSD	XTZUSDC	XTZUSDT	YFIAUD	YFIBTC	YFIETH
YFIEUR	YFIGBP	YFIUSD	YGGEUR	YGGUSD	ZECBTC
ZECEUR	ZECUSD	ZEREBROEUR	ZEREBROUSD	ZETAEUR	ZETAUSD
ZEUSEUR	ZEUSUSD	ZEXEUR	ZEXUSD	ZKEUR	ZKUSD
ZORAEUR	ZORAUSD	ZROEUR	ZROUSD	ZRXBTC	ZRXEUR
ZRXGBP	ZRXUSD				

Example Applications

The Kraken Crypto Price dataset enables you to accurately design strategies for Cryptocurrencies. Examples include the following strategies:

- Buy and hold
- Trading Cryptocurrency volatility and price action
- Allocating a small portion of your portfolio to Cryptocurrencies to hedge against inflation

Classic Algorithm Example

The following example algorithm buys and holds Bitcoin through the Kraken exchange:

C#

```
public class CoinAPIDataAlgorithm : QCAlgorithm
{
    private Crypto _btc;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        UniverseSettings.Asynchronous = true;
        SetCash(100000);
        // Kraken accepts both Cash and Margin account types. Select the
        // one you need for the best reality modeling.
        SetBrokerageModel(BrokerageName.Kraken, AccountType.Margin);
        // Add BTCUSD data.
        _btc = AddCrypto("BTCUSD", Resolution.Minute, Market.Kraken);
        // Historical data
        var history = History(_btc.Symbol, 30, Resolution.Daily);
        // Add a Crypto universe that selects trading pairs on Kraken.
        var universe = AddUniverse(CryptoUniverse.Kraken(SelectAssets));
        // Historical Universe data
        var universeHistory = History(universe, 30, Resolution.Daily);
        foreach (var universeDay in universeHistory)
        {
            foreach (CryptoUniverse universeItem in universeDay)
            {
                var symbol = universeItem.Symbol;
                var price = universeItem.Close;
            }
        }
    }

    private IEnumerable<Symbol> SelectAssets(IEnumerable<CryptoUniverse> universeDay)
    {
        // Filter for materially traded Crypto pairs with significant
        // size and dollar volume, assuming higher capital flow in for
        // higher return.
        return from universeItem in universeDay
                where universeItem.Volume >= 100m
                && universeItem.VolumeInUsd > 10000000m
                select universeItem.Symbol;
    }

    public override void OnData(Slice slice)
    {
        // Speculate-invest all available free cash on BTCUSD, repecting
        // the order quantity restrictions to avoid invalid orders.
        if (Portfolio.CashBook["BTC"].Amount != 0)
        {
            return;
        }
        // Define a buffer so that we can scale down the order size,
        // avoiding trading errors.
        var buffer = Math.Max(KrakenFeeModel.TakerTier1CryptoFee, Settings.FreePortfolioValuePercentage);
        var quantity = Portfolio.CashBook["USD"].Amount / _btc.AskPrice * (1-buffer);
        // Check if the quantity exceeds the minimum order size. On
        // Kraken, the minimum order size is measured in the base
        // currency (BTC in this case).
        if (Math.Abs(quantity) < _btc.SymbolProperties.MinimumOrderSize)
        {
            return;
        }
        MarketOrder(_btc, quantity);
    }
}
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of Crypto pairs on the Kraken exchange and then forms a equal-weighted portfolio of all the pairs in the universe:

```C#
public class CoinAPIDataAlgorithm : QCAlgorithm
{
    private List<string> _marketPairs;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        Settings.FreePortfolioValuePercentage = 0.05m;

        // Kraken accepts both Cash and Margin type account, select the one you need for the best reality modeling.
        SetBrokerageModel(BrokerageName.Kraken, AccountType.Margin);

        // Warm up the security with the last known price to avoid conversion error
        Settings.SeedInitialPrices = true;

        // Get the pairs on Kraken that have USD as the quote currency.
        _marketPairs = SymbolPropertiesDatabase.GetSymbolPropertiesList(Market.Kraken)
            .Where(x => x.Value.QuoteCurrency == AccountCurrency)
            .Select(x => x.Key.Symbol)
            .ToList();
        // Add Crypto Universe Selection that select crypto pairs in Kraken exchange
        AddUniverse(CryptoUniverse.Kraken(UniverseSelectionFilter));

        AddAlpha( new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromHours(4), 0.025, null)
);
        // Equally invest to evenly dissipate the capital concentration risk of inidividual crypto pair
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }

    private IEnumerable<Symbol> UniverseSelectionFilter(IEnumerable<CryptoUniverse> universeDay)
    {
        // Select pairs where the quote currency is USD.
        return universeDay.Where(c => _marketPairs.Contains(c.Symbol.Value))
            // Filter for materially traded crypto pairs with significant size and dollar volume, assuming higher
capital flow in for higher return
            .OrderBy(c => c.VolumeInUsd)
            .TakeLast(3)
            .Select(c => c.Symbol);
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach(var security in changes.AddedSecurities)
        {
            // Historical data
            var history = History(security.Symbol, 30, Resolution.Daily);
            Debug($"We got {history.Count()} items from our history request");
        }
    }
}
```

Research Example

The following example lists crypto-currency pairs with the greatest dollar volume in the Kraken exchange:

```C#
var qb = new QuantBook();

// Add Cryptocurrency pair
var symbol = qb.AddCrypto("BTCUSDT", market:Market.Kraken).Symbol;

// Historical data
var history = qb.History(symbol, 30, Resolution.Daily);
foreach (var bar in history)
{
    Console.WriteLine($"{bar.EndTime} {bar}");
}

// Add Crypto Universe Selection
IEnumerable<Symbol> UniverseSelectionFilter(IEnumerable<CryptoUniverse> universeDay)
{
    return universeDay
        .Where(x => x.VolumeInUsd != null && x.VolumeInUsd > 10000m)
        .OrderByDescending(x => x.VolumeInUsd)
        .Take(5)
        .Select(x => x.Symbol);
}
var universe = qb.AddUniverse(CryptoUniverse.Kraken(UniverseSelectionFilter));

// Historical Universe data
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var universeDay in universeHistory)
{
    Console.WriteLine($"=== {universeDay.First().EndTime} ===");
    foreach (CryptoUniverse universeItem in universeDay.OrderByDescending(x => x.VolumeInUsd))
    {
        Console.WriteLine($"{universeItem.Symbol}: {universeItem.VolumeInUsd}");
    }
}
```

Data Point Attributes

The Kraken Crypto Price dataset provides `TradeBar` , `QuoteBar` , `Tick` , and `CryptoUniverse` objects.

TradeBar Attributes

`TradeBar` objects have the following attributes:

QuoteBar Attributes

`QuoteBar` objects have the following attributes:

Tick Attributes

`Tick` objects have the following attributes:

CryptoUniverse Attributes

`CryptoUniverse` objects have the following attributes:

Datasets

Benzinga

Benzinga was founded by Jason Raznick in 2010 with goal of connecting the world with news, data, and education that makes the path to financial prosperity easier for everyone, everyday. Benzinga provides access to real-time news for individual investors.

Benzinga News Feed

Benzinga

Benzinga News Feed

Introduction

The Benzinga News Feed dataset by Benzinga tracks US Equity news releases. The data covers about 1,250 articles per day across 8,000 Equities, starts in January 2016, and is delivered on a second frequency. This dataset is created by structuring the content produced by Benzinga's editorial team.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Benzinga News Feed dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Benzinga was founded by Jason Raznick in 2010 with goal of connecting the world with news, data, and education that makes the path to financial prosperity easier for everyone, everyday. Benzinga provides access to real-time news for individual investors.

Getting Started

The following snippet demonstrates how to request data from the Benzinga News Feed dataset:

```C#
_symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
_datasetSymbol = AddData<BenzingaNews>(_symbol).Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	September 2017
Asset Coverage	8,000 Equities
Data Density	Sparse
Resolution	Second (1,250 Articles/Day)
Timezone	New York

Requesting Data

To add Benzinga News Feed data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```
C#
public class BenzingaNewsDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2021, 1, 1);
        SetEndDate(2021, 6, 1);
        SetCash(100000);

        _symbol = AddEquity("AAPL", Resolution.Minute).Symbol;
        _datasetSymbol = AddData<BenzingaNews>(_symbol).Symbol;
    }
}
```

Accessing Data

To get the current Benzinga News Feed data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var article = slice[_datasetSymbol];
        Log($"{_datasetSymbol} title at {slice.Time}: {article.Mentions}");
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<BenzingaNews>())
    {
        var datasetSymbol = kvp.Key;
        var article = kvp.Value;
        Log($"{datasetSymbol} title at {slice.Time}: {article.Title}");
    }
}
```

Historical Data

To get historical Benzinga News Feed data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
var history = History<BenzingaNews>(_datasetSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```
C#
RemoveSecurity(_datasetSymbol);
```

If you subscribe to Benzinga News Feed data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Benzinga News Feed enables you to accurately design strategies harnessing real-time news releases. Examples include the following strategies:

- Creating a dictionary of sentiment scores for various words and assigning a sentiment score to the content of each news release
- Calculating the sentiment of news releases with Natural Language Processing (NLP)
- Trading securities when their news releases that Benzinga tags with current buzzwords

Classic Algorithm Example

The following example algorithm parses the Benzinga news articles related to Apple. If the sentiment is positive, the algorithm buys Apple. Otherwise, it holds cash.

```C#
using System.Text.RegularExpressions;

public class BenzingaNewsDataAlgorithm : QCAlgorithm
{
    private Symbol _aapl;
    private Symbol _benzingaSymbol;
    private int _currentHoldings = 0;
    private int _targetHoldings = 0;
    //  A custom word-score map for calculating the total sentiment score
    private Dictionary<string, int> _wordScores = new Dictionary<string, int>(){
        {"good", 1}, {"great", 1}, {"best", 1}, {"growth", 1},
        {"bad", -1}, {"terrible", -1}, {"worst", -1}, {"loss", -1}
    };

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Requesting data to obtain the updated news for sentiment score calculation
        _aapl = AddEquity("AAPL", Resolution.Minute).Symbol;
        _benzingaSymbol = AddData<BenzingaNews>(_aapl).Symbol;

        // Historical data
        var history = History<BenzingaNews>(_benzingaSymbol, 14, Resolution.Daily);
        Debug($"We got {history.Count()} items from our history request");
    }

    public override void OnData(Slice slice)
    {
        if (slice.ContainsKey(_benzingaSymbol))
        {
            // Assign a sentiment score to the news article by specific word appearance scoring
            var contentWords = slice[_benzingaSymbol].Contents.ToLower();
            var score = 0;
            foreach (KeyValuePair<string, int> entry in _wordScores)
            {
                score += (Regex.Matches(contentWords, entry.Key).Count * entry.Value);
            }
            _targetHoldings = Convert.ToInt32(score > 0);
        }

        // Ensure we have AAPL data in the current Slice to avoid stale filling
        if (!(slice.ContainsKey(_aapl) && slice[_aapl] != null && !slice[_aapl].IsFillForward))
        {
            return;
        }

        // Buy or sell if the sentiment has changed from our current holdings
        if (_currentHoldings != _targetHoldings)
        {
            SetHoldings(_aapl, _targetHoldings);
            _currentHoldings = _targetHoldings;
        }
    }
}
```

Framework Algorithm Example

The following example algorithm parses the Benzinga news articles related to Apple. If the sentiment is positive, the algorithm buys Apple. Otherwise, it holds cash.

```C#
using System.Text.RegularExpressions;

public class BenzingaNewsDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        AddUniverseSelection(
```

```
        AddUniverseSelection(
            new ManualUniverseSelectionModel(
                QuantConnect.Symbol.Create("AAPL", SecurityType.Equity, Market.USA)
        ));

        AddAlpha(new BenzingaAlphaModel());

        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());

        AddRiskManagement(new NullRiskManagementModel());

        SetExecution(new ImmediateExecutionModel());
    }
}

public class BenzingaAlphaModel : AlphaModel
{
    private Dictionary<Symbol, SymbolData> _symbolDataBySymbol = new Dictionary<Symbol, SymbolData>();
    // A custom word-score map for calculating the total sentiment score
    private Dictionary<string, int> _wordScores = new Dictionary<string, int>(){
        {"good", 1}, {"great", 1}, {"best", 1}, {"growth", 1},
        {"bad", -1}, {"terrible", -1}, {"worst", -1}, {"loss", -1}
    };

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = new List<Insight>();

        foreach (var kvp in _symbolDataBySymbol)
        {
            var symbol = kvp.Key;
            var symbolData = kvp.Value;

            if (slice.ContainsKey(symbolData.benzingaSymbol))
            {
                // Assign a sentiment score to the news article by specific word appearance scoring
                var contentWords = slice[symbolData.benzingaSymbol].Contents.ToLower();
                var score = 0;
                foreach (var entry in _wordScores)
                {
                    score += (Regex.Matches(contentWords, entry.Key).Count * entry.Value);
                }
                symbolData.targetDirection = score > 0 ? InsightDirection.Up : InsightDirection.Flat;
            }


            // Ensure we have security data in the current Slice to avoid stale filling
            if (!(slice.ContainsKey(symbol) && slice[symbol] != null && !slice[symbol].IsFillForward))
            {
                continue;
            }

            // Buy or sell if the sentiment has changed from our current holdings
            if (symbolData.currentDirection != symbolData.targetDirection)
            {
                insights.Add(Insight.Price(symbol, TimeSpan.FromDays(14), symbolData.targetDirection));
                symbolData.currentDirection = symbolData.targetDirection;
            }
        }
        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            var symbol = security.Symbol;
            _symbolDataBySymbol.Add(symbol, new SymbolData(algorithm, symbol));
        }

        foreach (var security in changes.RemovedSecurities)
        {
            var symbol = security.Symbol;
            if (_symbolDataBySymbol.ContainsKey(symbol))
            {
                _symbolDataBySymbol[symbol].dispose();
                _symbolDataBySymbol.Remove(symbol);
            }
        }
    }
}
```

```
public class SymbolData
{
    private QCAlgorithm _algorithm;
    public Symbol benzingaSymbol;
    public InsightDirection currentDirection = InsightDirection.Flat;
    public InsightDirection targetDirection = InsightDirection.Flat;


    public SymbolData(QCAlgorithm algorithm, Symbol symbol)
    {
        _algorithm = algorithm;

        // Requesting data to obtain the updated news for sentiment score calculation
        benzingaSymbol = algorithm.AddData<BenzingaNews>(symbol).Symbol;

        // Historical data
        var history = algorithm.History<BenzingaNews>(benzingaSymbol, 14, Resolution.Daily);
        algorithm.Debug($"We got {history.Count()} items from our history request");
    }

    public void dispose()
    {
        // Unsubscribe from the Benzinga feed for this security to release computational resources
        _algorithm.RemoveSecurity(benzingaSymbol);
    }
}
```

Data Point Attributes

The Benzinga News Feed dataset provides `BenzingaNews` objects, which have the following attributes:

Datasets

Blockchain

Blockchain is a website that publishes data related to Bitcoin. It has been online since 2011 and publishes the Bitcoin Metadata history back to 2009.

Bitcoin Metadata

Blockchain

Bitcoin Metadata

Introduction

The Bitcoin Metadata dataset by Blockchain provides 23 fundamental metadata of Bitcoin directly fetched from the Bitcoin blockchain. The data starts in January 2009 and delivered on a daily frequency. This dataset contains mining statistics like hash rate and miner revenue; transaction metadata like transaction per block, transaction fee, and number of addresses; and blockchain metadata like blockchain size and block size.

For more information about the Bitcoin Metadata dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Blockchain is a website that publishes data related to Bitcoin. It has been online since 2011 and publishes the Bitcoin Metadata history back to 2009.

Getting Started

The following snippet demonstrates how to request data from the Bitcoin Metadata dataset:

```C#
_symbol = AddCrypto("BTCUSD", Resolution.Daily, Market.Bitfinex).Symbol;
_datasetSymbol = AddData<BitcoinMetadata>(_symbol).Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2009
Coverage	Bitcoin blockchain
Data Density	Regular
Resolution	Daily
Timezone	UTC

Requesting Data

To add Bitcoin Metadata data to your algorithm, call the `AddData` method with the BTCUSD `Symbol` . Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```
C#
public class BlockchainBitcoinMetadataAlgorithm: QCAlgorithm
{
    private Symbol _symbol, _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);

        _symbol = AddCrypto("BTCUSD", Resolution.Daily, Market.Bitfinex).Symbol;
        _datasetSymbol = AddData<BitcoinMetadata>(_symbol).Symbol;
    }
}
```

Accessing Data

To get the current Bitcoin Metadata data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var dataPoint = slice[_datasetSymbol];
        Log($"{_datasetSymbol} miner revenue at {slice.Time}: {dataPoint.MinersRevenue}");
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<BlockchainBitcoinData>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} miner revenue at {slice.Time}: {dataPoint.MinersRevenue}");
    }
}
```

Historical Data

To get historical Bitcoin Metadata data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
var history = History<BlockchainBitcoinData>(_datasetSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```
C#
RemoveSecurity(_datasetSymbol);
```

Example Applications

The Bitcoin Metadata dataset enables you to incorporate metadata from the Bitcoin blockchain into your strategies. Examples include the following strategies:

- Comparing mining and transaction statistics to provide insight on the supply-demand relationship of the Bitcoin blockchain service.
- Measuring the activity and popularity of the Bitcoin blockchain to predict the price movements of the Cryptocurrency.

Classic Algorithm Example

The following example algorithm tracks the transaction-to-hash-rate ratio of the Bitcoin network. The algorithm holds Bitcoin when the ratio increases. Otherwise, it holds dollars.

C#

```
public class BlockchainBitcoinMetadataAlgorithm : QCAlgorithm
{
    private Symbol _bitcoinMetadataSymbol, _btcSymbol;
    // Cache the last supply-demand ratio for comparison
    private decimal? _lastDemandSupply = null;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Request BTCUSD as the trading vehicle on Bitcoin Metadata
        _btcSymbol = AddCrypto("BTCUSD", Resolution.Minute, Market.Bitfinex).Symbol;
        // Request Bitcoin Metadata for trade signal generation
        _bitcoinMetadataSymbol = AddData<BitcoinMetadata>(_btcSymbol).Symbol;

        // Historical data
        var history = History(new[]{_bitcoinMetadataSymbol}, 60, Resolution.Daily);
        Debug($"We got {history.Count()} items from our history request for {_btcSymbol} Blockchain Bitcoin
Metadata");
    }

    public override void OnData(Slice slice)
    {
        // Trade only based on updated Bitcoin Metadata
        var data = slice.Get<BitcoinMetadata>();
        if (!data.IsNullOrEmpty())
        {
            // Calculate the supply-demand ratio to estimate the microeconomy structure of Bitcoin for scalp-trading
            // Transaction number as demand, hash production rate as supply
            if (data[_bitcoinMetadataSymbol].HashRate == 0)
            {
                return;
            }
            var currentDemandSupply = data[_bitcoinMetadataSymbol].NumberofTransactions /
data[_bitcoinMetadataSymbol].HashRate;

            // Comparing the average transaction-to-hash-rate ratio changes, buy Bitcoin if demand is higher than
supply, sell vice versa
            if (_lastDemandSupply != null && currentDemandSupply > _lastDemandSupply)
            {
                SetHoldings(_btcSymbol, 1);
            }
            else
            {
                SetHoldings(_btcSymbol, 0);
            }

            _lastDemandSupply = currentDemandSupply;
        }
    }
}
```

Framework Algorithm Example

The following example algorithm tracks the transaction-to-hash-rate ratio of the Bitcoin network. The algorithm holds Bitcoin when the ratio increases. Otherwise, it holds dollars.

C#

```
public class BlockchainBitcoinMetadataFrameworkAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Universe contains only BTCUSD as the trading vehicle on Bitcoin Metadata
        AddUniverseSelection(
            new ManualUniverseSelectionModel(
            QuantConnect.Symbol.Create("BTCUSD", SecurityType.Crypto, Market.Bitfinex)
        ));
        // Custom alpha model that emit insights based on Bitcoin Metadata
        AddAlpha(new BlockchainBitcoinMetadataAlphaModel());
        // Equally invest to evenly dissipate the capital concentration risk from non-sysmtematic risky events
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }
}

public class BlockchainBitcoinMetadataAlphaModel: AlphaModel
{
    private Dictionary<Symbol, Symbol> _bitcoinMetadataSymbolBySymbol = new Dictionary<Symbol, Symbol>();
    // Cache the last supply-demand ratio for comparison
    private Dictionary<Symbol, decimal> _lastDemandSupply = new Dictionary<Symbol, decimal>();

    public BlockchainBitcoinMetadataAlphaModel(){}

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = new List<Insight>();

        // Trade only based on updated Bitcoin Metadata
        var data = slice.Get<BitcoinMetadata>();
        if (!data.IsNullOrEmpty())
        {
            foreach(var kvp in _bitcoinMetadataSymbolBySymbol)
            {
                var symbol = kvp.Key;
                var bitcoinMetadataSymbol = kvp.Value;

                // Calculate the supply-demand ratio to estimate the microeconomy structure of the crypto pair for
scalp-trading
                // Transaction number as demand, hash production rate as supply
                if (data[bitcoinMetadataSymbol].HashRate == 0) continue;
                var currentDemandSupply = data[bitcoinMetadataSymbol].NumberofTransactions /
data[bitcoinMetadataSymbol].HashRate;

                // Comparing the average transaction-to-hash-rate ratio changes, buy coin if demand is higher than
supply
                if (_lastDemandSupply.ContainsKey(symbol) && currentDemandSupply > _lastDemandSupply[symbol])
                {
                    insights.Add(Insight.Price(symbol, TimeSpan.FromDays(1), InsightDirection.Up));
                }

                _lastDemandSupply[symbol] = currentDemandSupply;
            }
        }

        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            var symbol = security.Symbol;
            // Request Bitcoin Metadata for trade signal generation
            var bitcoinMetadataSymbol = algorithm.AddData<BitcoinMetadata>(symbol).Symbol;

            _bitcoinMetadataSymbolBySymbol.Add(symbol, bitcoinMetadataSymbol);

            // Historical data
            var history = algorithm.History(new[]{bitcoinMetadataSymbol}, 60, Resolution.Daily);
        }
    }
}
```

Data Point Attributes

The Bitcoin Metadata dataset provides `BitcoinMetadata` objects, which have the following attributes:

Datasets

Brain

Brain is a Research Company that creates proprietary datasets and algorithms for investment strategies, combining experience in financial markets with strong competencies in Statistics, Machine Learning, and Natural Language Processing. The founders share a common academic background of research in Physics as well as extensive experience in Financial markets.

Brain Language Metrics on Company Filings

Brain ML Stock Ranking

Brain Sentiment Indicator

Brain Wikipedia Page Views

Brain

Brain Language Metrics on Company Filings

Introduction

The Brain Language Metrics on Company Filings dataset provides the results of an NLP system that monitors several language metrics on 10-K and 10-Q company reports for US Equities. The data covers 5,000 US Equities, starts in January 2010, and is delivered on a daily frequency. The dataset is made of two parts; the first one includes the language metrics of the most recent 10-K or 10-Q report for each firm, namely:

1. Financial sentiment
2. Percentage of words belonging to financial domain classified by language types (e.g. “litigious” or “constraining” language)
3. Readability score
4. Lexical metrics such as lexical density and richness
5. Text statistics such as the report length and the average sentence length

The second part includes the differences between the two most recent 10-Ks or 10-Qs reports of the same period for each company, namely:

1. Difference of the various language metrics (e.g. delta sentiment, delta readability score, delta percentage of a specific language type etc.)
2. Similarity metrics between documents, also with respect to a specific language type (for example similarity with respect to “litigious” language or “uncertainty” language)

The analysis is available for the whole report and for specific sections of the report (e.g. Risk Factors and MD&A).

For more information, refer to Brain's summary paper .

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Brain Language Metrics on Company Filings dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Brain is a Research Company that creates proprietary datasets and algorithms for investment strategies, combining experience in financial markets with strong competencies in Statistics, Machine Learning, and Natural Language Processing. The founders share a common academic background of research in Physics as well as extensive experience in Financial markets.

Getting Started

The following snippet demonstrates how to request data from the Brain Language Metrics on Company Filings dataset:

```
C#
_symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
_dataset10KSymbol = AddData<BrainCompanyFilingLanguageMetrics10K>(_symbol).Symbol;
_datasetAllSymbol = AddData<BrainCompanyFilingLanguageMetricsAll>(_symbol).Symbol;

_universe10k = AddUniverse<BrainCompanyFilingLanguageMetricsUniverse10K>(UniverseSelection);
_universeAll = AddUniverse<BrainCompanyFilingLanguageMetricsUniverseAll>(UniverseSelection);
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2010
Asset Coverage*	5,000 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

The coverage includes all assets since the start date. It increases over time.

Requesting Data

To add Brain Language Metrics on Company Filings data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```
C#
public class BrainCompanyFilingNLPDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _dataset10KSymbol, _datasetAllSymbol;

    public override void Initialize()
    {
        SetStartDate(2010, 1, 1);
        SetEndDate(2021, 7, 8);
        SetCash(100000);

        _symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _dataset10KSymbol= AddData<BrainCompanyFilingLanguageMetrics10K>(_symbol).Symbol;
        _datasetAllSymbol= AddData<BrainCompanyFilingLanguageMetricsAll>(_symbol).Symbol;
    }
}
```

Accessing Data

To get the current Brain Language Metrics on Company Filings data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_dataset10KSymbol))
    {
        var dataPoint = slice[_dataset10KSymbol];
        Log($"{_dataset10KSymbol} report sentiment at {slice.Time}: {dataPoint.ReportSentiment.Sentiment}");
    }

    if (slice.ContainsKey(_datasetAllSymbol))
    {
        var dataPoint = slice[_datasetAllSymbol];
        Log($"{_datasetAllSymbol} report sentiment at {slice.Time}: {dataPoint.ReportSentiment.Sentiment}");
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<BrainCompanyFilingLanguageMetrics10K>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} report sentiment at {slice.Time}: {dataPoint.ReportSentiment.Sentiment}");
    }

    foreach (var kvp in slice.Get<BrainCompanyFilingLanguageMetricsAll>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} report sentiment at {slice.Time}: {dataPoint.ReportSentiment.Sentiment}");
    }
}
```

Historical Data

To get historical Brain Language Metrics on Company Filings data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
// Dataset objects
var tenKHistory = History<BrainCompanyFilingLanguageMetrics10K>(_dataset10KSymbol, 100, Resolution.Daily);
var allHistory = History<BrainCompanyFilingLanguageMetricsAll>(_datasetAllSymbol, 100, Resolution.Daily);

// Slice objects
var history = History(new[] {_dataset10KSymbol, _datasetAllSymbol}, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on Brain Language Metrics on Company Filings data, call the `AddUniverse` method with the `BrainCompanyFilingLanguageMetricsUniverseAll` class or the `BrainCompanyFilingLanguageMetricsUniverse10K` class and a selection function.

```C#
private Universe _universe;
public override void Initialize()
{
    _universe = AddUniverse<BrainCompanyFilingLanguageMetricsUniverseAll>(altCoarse =>
    {
        return from d in altCoarse.OfType<BrainCompanyFilingLanguageMetricsUniverseAll>()
            where d.ReportSentiment.Sentiment > 0m &&
                      d.ManagementDiscussionAnalyasisOfFinancialConditionAndResultsOfOperations.Sentiment > 0m
            select d.Symbol;
    });
}
```

For more information about dynamic universes, see Universes .

Universe History

You can get historical universe data in an algorithm and in the Research Environment.

Historical Universe Data in Algorithms

To get historical universe data in an algorithm, call the `History` method with the `Universe` object and the lookback period. If there is no data in the period you request, the history result is empty.

```C#
var universeHistory = History(_universe, 30, Resolution.Daily);
foreach (var universeDay in universeHistory)
{
    foreach (BrainCompanyFilingLanguageMetricsUniverse10K languageMetrics in universeDay)
    {
        Log($"{languageMetrics.Symbol} sentiment at {languageMetrics.EndTime}:
{languageMetrics.ReportSentiment.Sentiment}");
    }
}
```

Historical Universe Data in Research

To get historical universe data in research, call the `UniverseHistory` method with the `Universe` object, a start date, and an end date. This method returns the filtered universe. If there is no data in the period you request, the history result is empty.

```C#
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var universeDay in universeHistory)
{
    foreach (BrainCompanyFilingLanguageMetricsUniverse10K languageMetrics in universeDay)
    {
        Console.WriteLine($"{languageMetrics.Symbol} sentiment at {languageMetrics.EndTime}:
{languageMetrics.ReportSentiment.Sentiment}");
    }
}
```

You can call the `History` method in Research.

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_dataset10KSymbol);
RemoveSecurity(_datasetAllSymbol);
```

If you subscribe to Brain Language Metrics on Company Filings data for assets in a dynamic universe, remove the dataset

subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Brain Language Metrics on Company Filings dataset enables you to test strategies using language metrics and their differences gathered from 10K and 10Q reports. Examples include the following strategies:

- Using the similarity among reports to determine position sizing of securities. Some examples are discussed in Lazy Prices, Cohen et al. 2018 and The Positive Similarity of Company Filings and the Cross-section of Stock Returns, M. Padyšák 2020 .
- Using the sentiment of the latest report to determine the portfolio allocation to give to each security in the universe.
- Using levels of uncertainty, readability, or litigious language in the report to determine position sizing of securities.

Classic Algorithm Example

The following example algorithm buys Apple when the sentiment of their 10K report is positive. Otherwise, it holds cash.

```C#
public class BrainCompanyFilingNLPDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Symbol _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Requesting data -- we aim to obtain a sentiment score from the company filings
        // Combining both fundamental and sentiment factor, as well as past performance and future provision
        _symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _datasetSymbol = AddData<BrainCompanyFilingLanguageMetrics10K>(_symbol).Symbol;

        // Historical data
        var history = History<BrainCompanyFilingLanguageMetrics10K>(_datasetSymbol, 365, Resolution.Daily);
        Debug($"We got {history.Count()} items from our history request for {_datasetSymbol}");
    }

    public override void OnData(Slice slice)
    {
        // Trade base on the updated report sentiment
        if (slice.ContainsKey(_datasetSymbol))
        {
            // Buy for a positive sentiment score for the positive return projection
            var sentiment = slice[_datasetSymbol].ReportSentiment.Sentiment;
            SetHoldings(_symbol, sentiment > 0 ? 1 : 0);
        }
    }
}
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of US Equities that have positive sentiment in their 10K report and then forms an equal-weighted portfolio:

C#

```
public class BrainCompanyFilingNLPDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        Settings.FreePortfolioValuePercentage = 0.1m;
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;

        // Filter based on company filing data
        UniverseSettings.Resolution = Resolution.Daily;
        AddUniverse<BrainCompanyFilingLanguageMetricsUniverse10K>(altCoarse =>
        {
            // Filter for the stocks with positive sentiment score for the positive return projection
            // Combining both fundamental and sentiment factor, as well as past performance and future provision
            return from d in altCoarse.OfType<BrainCompanyFilingLanguageMetricsUniverse10K>()
                where d.ReportSentiment.Sentiment > 0m
                select d.Symbol;
        });

        AddAlpha(new BrainCompanyFilingNLPAlphaModel());

        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());

        AddRiskManagement(new NullRiskManagementModel());

        SetExecution(new ImmediateExecutionModel());
    }
}

public class BrainCompanyFilingNLPAlphaModel : AlphaModel
{
    private List<Security> _securities = new();

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        // Signal to invest in the selected universe, which are expected to have a positive return due to positive
fundamentals and future prospect
        return _securities
            .Where(security => security.Price != 0 && security.IsTradable)
            .Select(security => Insight.Price(security.Symbol, TimeSpan.FromDays(1), InsightDirection.Up));
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes) {
        foreach (var security in changes.AddedSecurities)
        {
            _securities.Add(security);
        }
        foreach (var security in changes.RemovedSecurities)
        {
            if (_securities.Contains(security))
            {
                _securities.Remove(security);
            }
        }
    }
}
```

Research Example

The following example lists US Equities having the highest 2-day rank.

```
C#
#r "../QuantConnect.DataSource.BrainSentiment.dll"
using QuantConnect.DataSource;

var qb = new QuantBook();

// Requesting data
var aapl = qb.AddEquity("AAPL", Resolution.Daily).Symbol;
var symbol = qb.AddData<BrainCompanyFilingLanguageMetrics10K>(aapl).Symbol;

// Historical data
var history = qb.History<BrainCompanyFilingLanguageMetrics10K>(symbol, 180, Resolution.Daily);
foreach (BrainCompanyFilingLanguageMetrics10K languageMetrics in history)
{
    Console.WriteLine($"{languageMetrics} at {languageMetrics.EndTime}");
}

// Add Universe Selection
IEnumerable<Symbol> UniverseSelection(IEnumerable<BaseData> altCoarse)
{
    var symbols = (from d in altCoarse.OfType<BrainCompanyFilingLanguageMetricsUniverse10K>()
        orderby d.ReportSentiment.Sentiment descending select d.Symbol).Take(10);
    return symbols;
}
var universe = qb.AddUniverse<BrainCompanyFilingLanguageMetricsUniverse10K>(UniverseSelection);

// Historical Universe data
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-10), qb.Time);
foreach (var universeDay in universeHistory)
{
    foreach (BrainCompanyFilingLanguageMetricsUniverse10K languageMetrics in universeDay)
    {
        Console.WriteLine($"{languageMetrics.Symbol} sentiment at {languageMetrics.EndTime}:
{languageMetrics.ReportSentiment.Sentiment}");
    }
}
```

Data Point Attributes

The Brain Language Metrics on Company Filings dataset provides `BrainCompanyFilingLanguageMetrics` and `BrainCompanyFilingLanguageMetricsUniverse` objects.

BrainCompanyFilingLanguageMetrics Attributes

`BrainCompanyFilingLanguageMetrics` objects have the following attributes:

BrainCompanyFilingLanguageMetricsUniverse Attributes

`BrainCompanyFilingLanguageMetricsUniverse` objects have the following attributes:

Brain

Brain ML Stock Ranking

Introduction

The Brain ML Stock Ranking dataset by Brain generates a daily ranking for US Equities based on their predicted ranking of future returns relative to the universe median across four-time horizons: next 2, 3, 5, 10, and 21 days (one trading month). The data covers 1,000 US Equities (universe updated yearly by including the largest 1,000 US companies of the previous year), starts in January 2010, and is delivered on a daily frequency. This dataset is created by a voting scheme of machine learning classifiers that non-linearly combine a variety of features with a series of techniques aimed at mitigating the well-known overfitting problem for financial data with a low signal-to-noise ratio. Examples of features are time-varying stock-specific features like price and volume-related metrics or fundamentals; time-fixed stock-specific features like the sector and other database information; market regime features such as volatility and other financial stress indicators; calendar features representing possible anomalies, for example, the month of the year.

More precisely the ML Stock Ranking score is related to the confidence of a Machine Learning classifier in predicting top or bottom quintile returns for the next N trading days (e.g. next 21 trading days) for a stock with the respect to the median of the universe and ranges from -1 to +1.

A negative score means that the system is more confident that the stock belongs to the lower returns quintile, a positive score means that the system is more confident that the stock belongs to the higher returns quintile. It is important to note that the score has a meaning only if used to compare different stocks to perform a ranking.

Typical use is to download the score for a large stock universe for a given day, e.g. 500 stocks or the full universe of 1000 stocks, order the stocks by mlAlpha score and go long the top K stocks, or build a long-short strategy going long the top K and short the bottom K stocks.

For more information, refer to Brain's summary paper .

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Brain ML Stock Ranking dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Brain is a Research Company that creates proprietary datasets and algorithms for investment strategies, combining experience in financial markets with strong competencies in Statistics, Machine Learning, and Natural Language Processing. The founders share a common academic background of research in Physics as well as extensive experience in Financial markets.

Getting Started

The following snippet demonstrates how to request data from the Brain ML Stock Ranking dataset:

C#

```
_symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
_datasetSymbol = AddData<BrainStockRanking2Day>(_symbol).Symbol;

_universe = AddUniverse<BrainStockRankingUniverse>(UniverseSelection);
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2010
Asset Coverage*	1,000 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

The coverage includes all assets since the start date. It increases over time.

Requesting Data

To add Brain ML Stock Ranking data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```
C#
public class BrainMLRankingDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _2DaySymbol, _3DaySymbol, _5DaySymbol, _10DaySymbol, _monthSymbol;

    public override void Initialize()
    {
        SetStartDate(2021, 1, 1);
        SetEndDate(2021, 7, 8);
        SetCash(100000);

        _symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _2DaySymbol = AddData<BrainStockRanking2Day>(_symbol).Symbol;
        _3DaySymbol = AddData<BrainStockRanking3Day>(_symbol).Symbol;
        _5DaySymbol = AddData<BrainStockRanking5Day>(_symbol).Symbol;
        _10DaySymbol = AddData<BrainStockRanking10Day>(_symbol).Symbol;
        _monthSymbol = AddData<BrainStockRanking21Day>(_symbol).Symbol;
    }
}
```

Accessing Data

To get the current Brain ML Stock Ranking data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_2DaySymbol))
    {
        var dataPoint = slice[_2DaySymbol];
        Log($"{_2DaySymbol} rank at {slice.Time}: {dataPoint.Rank}");
    }

    if (slice.ContainsKey(_3DaySymbol))
    {
        var dataPoint = slice[_3DaySymbol];
        Log($"{_3DaySymbol} rank at {slice.Time}: {dataPoint.Rank}");
    }

    if (slice.ContainsKey(_5DaySymbol))
    {
        var dataPoint = slice[_5DaySymbol];
        Log($"{_5DaySymbol} rank at {slice.Time}: {dataPoint.Rank}");
    }

    if (slice.ContainsKey(_10DaySymbol))
    {
        var dataPoint = slice[_10DaySymbol];
        Log($"{_10DaySymbol} rank at {slice.Time}: {dataPoint.Rank}");
    }

    if (slice.ContainsKey(_monthSymbol))
    {
        var dataPoint = slice[_monthSymbol];
        Log($"{_monthSymbol} rank at {slice.Time}: {dataPoint.Rank}");
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<BrainStockRanking2Day>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} rank at {slice.Time}: {dataPoint.Rank}");
    }

    foreach (var kvp in slice.Get<BrainStockRanking3Day>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} rank at {slice.Time}: {dataPoint.Rank}");
    }

    foreach (var kvp in slice.Get<BrainStockRanking5Day>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} rank at {slice.Time}: {dataPoint.Rank}");
    }

    foreach (var kvp in slice.Get<BrainStockRanking10Day>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} rank at {slice.Time}: {dataPoint.Rank}");
    }

    foreach (var kvp in slice.Get<BrainStockRanking21Day>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} rank at {slice.Time}: {dataPoint.Rank}");
    }
}
```

Historical Data

To get historical Brain ML Stock Ranking data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#

// Dataset objects
var twoDayHistory = History<BrainStockRanking2Day>(_2DaySymbol, 100, Resolution.Daily);
var threeDayHistory = History<BrainStockRanking3Day>(_3DaySymbol, 100, Resolution.Daily);
var fiveDayHistory = History<BrainStockRanking5Day>(_5DaySymbol, 100, Resolution.Daily);
var tenDayHistory = History<BrainStockRanking10Day>(_10DaySymbol, 100, Resolution.Daily);
var monthHistory = History<BrainStockRanking21Day>(_monthSymbol, 100, Resolution.Daily);

// Slice objects
var history = History(new[] {_2DaySymbol, _3DaySymbol, _5DaySymbol,
                             _10DaySymbol, _monthSymbol}, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on Brain ML Stock Ranking data, call the `AddUniverse` method with the `BrainStockRankingUniverse` class and a selection function.

```
C#

private Universe _universe;
public override void Initialize()
{
    _universe = AddUniverse<BrainStockRankingUniverse>(altCoarse =>
    {
        return from d in altCoarse.OfType<BrainStockRankingUniverse>()
            where d.Rank2Days > 0m && d.Rank3Days > 0m  && d.Rank5Days > 0m
            select d.Symbol;
    })
};
```

For more information about dynamic universes, see Universes .

Universe History

You can get historical universe data in an algorithm and in the Research Environment.

Historical Universe Data in Algorithms

To get historical universe data in an algorithm, call the `History` method with the `Universe` object and the lookback period. If there is no data in the period you request, the history result is empty.

```
C#

var universeHistory = History(_universe, 30, Resolution.Daily);
foreach (var ranks in universeHistory)
{
    foreach (BrainStockRankingUniverse rank in ranks)
    {
        Log($"{rank.Symbol} 2-day rank at {rank.EndTime}: {rank.Rank2Days}");
    }
}
```

Historical Universe Data in Research

To get historical universe data in research, call the `UniverseHistory` method with the `Universe` object, a start date, and an end date. This method returns the filtered universe. If there is no data in the period you request, the history result is empty.

```
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var ranks in universeHistory)
{
    foreach (BrainStockRankingUniverse rank in ranks)
    {
        Console.WriteLine($"{rank.Symbol} 2-day rank at {rank.EndTime}: {rank.Rank2Days}");
    }
}
```

You can call the `History` method in Research.

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```
RemoveSecurity(_2DaySymbol);
RemoveSecurity(_3DaySymbol);
RemoveSecurity(_5DaySymbol);
RemoveSecurity(_10DaySymbol);
RemoveSecurity(_monthSymbol);
```

If you subscribe to Brain ML Stock Ranking data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Brain ML Stock Ranking dataset enables you to test strategies using the machine learning ranking provided by Brain. Examples include the following strategies:

- Constructing a portfolio of securities with each security's weight in the portfolio reflecting its Brain ML Stock Ranking
- Buying stocks with the largest Brain ML Stock Ranking
- Building a market-neutral strategy based on the top N and bottom N stocks in the Brain ML Stock Ranking

Classic Algorithm Example

The following example algorithm constructs a portfolio where the weight of each security in the portfolio is scaled based on its Brain ML Ranking. It gives a larger allocation to securities that have a higher Brain ML Ranking.

```C#
public class BrainMLRankingDataAlgorithm : QCAlgorithm
{
    private Dictionary<Symbol, Symbol> _symbolByDatasetSymbol = new Dictionary<Symbol, Symbol>();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // We cherry picked 5 largest stocks, high trading volume provides better information and credibility for ML
ranking
        var tickers = new List<string>() {"AAPL", "TSLA", "MSFT", "F", "KO"};
        foreach (var ticker in tickers)
        {
            // Requesting data to get 2 days estimated relative ranking
            var symbol = AddEquity(ticker, Resolution.Daily).Symbol;
            var datasetSymbol = AddData<BrainStockRanking2Day>(symbol).Symbol;
            _symbolByDatasetSymbol.Add(datasetSymbol, symbol);

            // Historical data
            var history = History<BrainStockRanking2Day>(datasetSymbol, 365, Resolution.Daily);
            Debug($"We got {history.Count()} items from our history request for {symbol}");
        }
    }

    public override void OnData(Slice slice)
    {
        // Collect rankings for all symbols for ranking them
        var points = slice.Get<BrainStockRanking2Day>();
        if (points == null)
        {
            return;
        }
        var symbols = new List<Symbol>();
        var ranks = new List<decimal>();
        foreach (var point in points.Values)
        {
            symbols.Add(_symbolByDatasetSymbol[point.Symbol]);
            ranks.Add(point.Rank);
        }

        // Rank each symbol's Brain ML ranking relative to the other symbols for positional sizing
        if (ranks.Count() == 0) return;
        var sortedRanksTemp = new List<decimal>(ranks);
        sortedRanksTemp.Sort();
        var sortedRanks = new List<decimal>();
        for (var i = 0; i < symbols.Count(); i++)
        {
            sortedRanks.Add(sortedRanksTemp.IndexOf(ranks[i]) + 1);
        }

        // Place orders, the better the rank, the higher the estimated return and hence weight
        for (var i = 0; i < symbols.Count(); i++)
        {
            var rank = sortedRanks[i];
            var weight = rank / sortedRanks.Sum();
            SetHoldings(symbols[i], weight);
        }
    }
}
```

Framework Algorithm Example

The following example algorithm constructs a portfolio where the weight of each security in the portfolio is scaled based on its Brain ML Ranking. It gives a larger allocation to securities that have a higher Brain ML Ranking.

```C#
public class BrainMLRankingDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
```

```
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Select based on ML ranking data
        AddUniverse<BrainStockRankingUniverse>(altCoarse =>
        {
            // Select the ones that expected to out-perform the median of the whole market in 2-3 days
            return from d in altCoarse.OfType<BrainStockRankingUniverse>()
                where d.Rank2Days > 0.2m && d.Rank3Days > 0.2m
                select d.Symbol;
        });

        AddAlpha(new BrainMLRankingAlphaModel());

        SetPortfolioConstruction(new InsightWeightingPortfolioConstructionModel());

        AddRiskManagement(new NullRiskManagementModel());

        SetExecution(new ImmediateExecutionModel());
    }
}


public class BrainMLRankingAlphaModel : AlphaModel
{
    private Dictionary<Symbol, SymbolData> _symbolDataBySymbol = new Dictionary<Symbol, SymbolData>();
    private Dictionary<Symbol, Symbol> _symbolByDatasetSymbol = new Dictionary<Symbol, Symbol>();

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = new List<Insight>();

        // Collect rankings for all selected symbols for ranking them
        var points = slice.Get<BrainStockRanking2Day>();
        if (points == null)
        {
            return insights;
        }
        var symbols = new List<Symbol>();
        var ranks = new List<decimal>();
        foreach (var point in points.Values)
        {
            symbols.Add(_symbolByDatasetSymbol[point.Symbol]);
            ranks.Add(point.Rank);
        }

        // Rank each symbol's Brain ML ranking relative to the other symbols for positional sizing
        if (ranks.Count() == 0)
        {
            return insights;
        }
        var sortedRanksTemp = new List<decimal>(ranks);
        sortedRanksTemp.Sort();
        var sortedRanks = new List<decimal>();
        for (var i = 0; i < symbols.Count(); i++)
        {
            sortedRanks.Add(sortedRanksTemp.IndexOf(ranks[i]) + 1);
        }

        // Place orders according to the ML ranking, the better the rank, the higher the estimated return and hence
weight
        for (var i = 0; i < symbols.Count(); i++)
        {
            var rank = sortedRanks[i];
            var weight = rank / sortedRanks.Sum();
            insights.Add(Insight.Price(symbols[i], TimeSpan.FromDays(7), InsightDirection.Up, null, null, null,
(double)weight));
        }

        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            var symbol = security.Symbol;
            var symbolData = new SymbolData(algorithm, symbol);
            _symbolDataBySymbol.Add(symbol, symbolData);
            _symbolByDatasetSymbol.Add(symbolData.datasetSymbol, symbol);
        }
```

```
        foreach (var security in changes.RemovedSecurities)
        {
            var symbol = security.Symbol;
            if (_symbolDataBySymbol.ContainsKey(symbol))
            {
                _symbolDataBySymbol[symbol].dispose();
                _symbolDataBySymbol.Remove(symbol);
            }

            foreach (var entry in _symbolByDatasetSymbol)
            {
                if (entry.Value == symbol)
                {
                    _symbolByDatasetSymbol.Remove(entry.Key);
                }
            }
        }
    }
}


public class SymbolData
{
    private QCAlgorithm _algorithm;
    public Symbol datasetSymbol;

    public SymbolData(QCAlgorithm algorithm, Symbol symbol)
    {
        _algorithm = algorithm;

        // Requesting data to get 2 days estimated relative ranking
        datasetSymbol = algorithm.AddData<BrainStockRanking2Day>(symbol).Symbol;

        // Historical data
        var history = algorithm.History<BrainStockRanking2Day>(datasetSymbol, 365, Resolution.Daily);
        algorithm.Debug($"We got {history.Count()} items from our history request for {symbol}");
    }

    public void dispose()
    {
        // Unsubscribe from the Brain ML Ranking feed for this security to release computation resources
        _algorithm.RemoveSecurity(datasetSymbol);
    }
}
```

Research Example

The following example lists US Equities having the highest 2-day rank.

C#

```
#r "../QuantConnect.DataSource.BrainSentiment.dll"
using QuantConnect.DataSource;

var qb = new QuantBook();

// Requesting data
var aapl = qb.AddEquity("AAPL", Resolution.Daily).Symbol;
var symbol = qb.AddData<BrainStockRanking2Day>(aapl).Symbol;

// Historical data
var history = qb.History<BrainStockRanking2Day>(symbol, 30, Resolution.Daily);
foreach (BrainStockRanking2Day rank in history)
{
    Console.WriteLine($"{rank} at {rank.EndTime}");
}

// Add Universe Selection
IEnumerable<Symbol> UniverseSelection(IEnumerable<BaseData> altCoarse)
{
    return (from d in altCoarse.OfType<BrainStockRankingUniverse>()
        orderby d.Rank2Days descending select d.Symbol).Take(10);
}
var universe = qb.AddUniverse<BrainStockRankingUniverse>(UniverseSelection);

// Historical Universe data
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-5), qb.Time);
foreach (var ranks in universeHistory)
{
    foreach (BrainStockRankingUniverse rank in ranks)
    {
        Console.WriteLine($"{rank.Symbol} 2-day rank at {rank.EndTime}: {rank.Rank2Days}");
    }
}
```

Data Point Attributes

The Brain ML Stock Ranking dataset provides `BrainStockRankingBase` and `BrainStockRankingUniverse` objects.

BrainStockRankingBase Attributes

`BrainStockRankingBase` objects have the following attributes:

BrainStockRankingUniverse Attributes

`BrainStockRankingUniverse` objects have the following attributes:

Brain

Brain Sentiment Indicator

Introduction

The Brain Sentiment Indicator dataset by Brain tracks the public sentiment around US Equities. The data covers 4,500 US Equities, starts in August 2016, and is delivered on a daily frequency. This dataset is created by analyzing financial news using Natural Language Processing techniques while taking into account the similarity and repetition of news on the same topic. The sentiment score assigned to each stock ranges from -1 (most negative) to +1 (most positive). The sentiment score corresponds to the average sentiment for each piece of news. The score is updated daily and is available on two time scales: 7 days and 30 days. For more information, see Brain's summary paper .

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Brain Sentiment Indicator dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Brain is a Research Company that creates proprietary datasets and algorithms for investment strategies, combining experience in financial markets with strong competencies in Statistics, Machine Learning, and Natural Language Processing. The founders share a common academic background of research in Physics as well as extensive experience in Financial markets.

Getting Started

The following snippet demonstrates how to request data from the Brain Sentiment Indicator dataset:

```C#
_symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
_dataset7DaySymbol = AddData<BrainSentimentIndicator7Day>(_symbol).Symbol;
_dataset30DaySymbol = AddData<BrainSentimentIndicator30Day>(_symbol).Symbol;

_universe = AddUniverse<BrainSentimentIndicatorUniverse>(UniverseSelection);
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	August 2016
Asset Coverage*	4,500 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

The coverage includes all assets since the start date. It increases over time.

Requesting Data

To add Brain Sentiment Indicator data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```
C#
public class BrainSentimentDataAlgorithm : QCAlgorithm
{
    private Symbol _dataset7DaySymbol, _dataset30DaySymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2021, 7, 8);
        SetCash(100000);

        var symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _dataset7DaySymbol = AddData<BrainSentimentIndicator7Day>(symbol).Symbol;
        _dataset30DaySymbol = AddData<BrainSentimentIndicator30Day>(symbol).Symbol;
    }
}
```

Accessing Data

To get the current Brain Sentiment Indicator data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_dataset7DaySymbol))
    {
        var dataPoint = slice[_dataset7DaySymbol];
        Log($"{_dataset7DaySymbol} sentiment at {slice.Time}: {dataPoint.Sentiment}");
    }

    if (slice.ContainsKey(_dataset30DaySymbol))
    {
        var dataPoint = slice[_dataset30DaySymbol];
        Log($"{_dataset30DaySymbol} sentiment at {slice.Time}: {dataPoint.Sentiment}");
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<BrainSentimentIndicator7Day>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} sentiment at {slice.Time}: {dataPoint.Sentiment}");
    }

    foreach (var kvp in slice.Get<BrainSentimentIndicator30Day>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} sentiment at {slice.Time}: {dataPoint.Sentiment}");
    }
}
```

Historical Data

To get historical Brain Sentiment Indicator data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
// Dataset objects
var weekHistory = History<BrainSentimentIndicator7Day>(_dataset7DaySymbol, 100, Resolution.Daily);
var monthHistory = History<BrainSentimentIndicator30Day>(_dataset30DaySymbol, 100, Resolution.Daily);

// Slice objects
var history = History(new[] {_dataset7DaySymbol, _dataset30DaySymbol}, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on Brain Sentiment Indicator data, call the `AddUniverse` method with the `BrainSentimentIndicatorUniverse` class and a selection function.

```
C#
private Universe _universe;
public override void Initialize()
{
    _universe = AddUniverse<BrainSentimentIndicatorUniverse>(altCoarse=>
    {
        return from d in altCoarse.OfType<BrainSentimentIndicatorUniverse>()
            where d.TotalArticleMentions7Days > 0m && d.Sentiment7Days > 0m
            select d.Symbol;
    });
}
```

The Brain Sentiment Indicator universe runs at 7 AM Eastern Time (ET) in live trading. For more information about dynamic universes, see Universes .

Universe History

You can get historical universe data in an algorithm and in the Research Environment.

Historical Universe Data in Algorithms

To get historical universe data in an algorithm, call the `History` method with the `Universe` object and the lookback period. If there is no data in the period you request, the history result is empty.

```
C#
var universeHistory = History(_universe, 30, Resolution.Daily);
foreach (var sentiments in universeHistory)
{
    foreach (BrainSentimentIndicatorUniverse sentiment in sentiments)
    {
        Log($"{sentiment.Symbol} 7-day sentiment at {sentiment.EndTime}: {sentiment.Sentiment7Days}");
    }
}
```

Historical Universe Data in Research

To get historical universe data in research, call the `UniverseHistory` method with the `Universe` object, a start date, and an end date. This method returns the filtered universe. If there is no data in the period you request, the history result is empty.

```
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var sentiments in universeHistory)
{
    foreach (BrainSentimentIndicatorUniverse sentiment in sentiments)
    {
        Console.WriteLine($"{sentiment.Symbol} 7-day sentiment at {sentiment.EndTime}: {sentiment.Sentiment7Days}");
    }
}
```

C#

You can call the `History` method in Research.

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```
RemoveSecurity(_dataset7DaySymbol);
RemoveSecurity(_dataset30DaySymbol);
```

C#

If you subscribe to Brain Sentiment Indicator data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Brain Sentiment Indicator dataset enables you to incorporate sentiment from financial news sources into your strategies. Examples include the following strategies:

- Buying when the public sentiment for a security is increasing
- Short selling when the public sentiment for a security is decreasing
- Scaling the position sizing of securities based on how many times they are mentioned in financial news articles
- Sector rotation based on news sentiment

Classic Algorithm Example

The following example algorithm buys Apple when the 30-day Brain Sentiment indicator increases. Otherwise, it remains in cash.

```C#
public class BrainSentimentDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Symbol _datasetSymbol;
    private decimal? _latestSentimentValue = null;
    private int _targetHoldings = 0;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Requesting the processed longer term (30-day) sentiment score data for sentiment trading
        _symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _datasetSymbol = AddData<BrainSentimentIndicator30Day>(_symbol).Symbol;

        // Historical data
        var history = History<BrainSentimentIndicator30Day>(_datasetSymbol, 100, Resolution.Daily);
        Debug($"We got {history.Count()} items from our history request for {_datasetSymbol}");

        // Warm up historical sentiment values, cache for comparing last sentiment score to trade, making it
immediately tradable signal
        var previousSentimentValues = history.Select(x => x.Sentiment);
        foreach (var sentiment in previousSentimentValues)
        {
            Update(sentiment);
        }
    }


    public void Update(decimal sentiment)
    {
        // Comparing the last sentiment score and decide to buy if the sentiment increases to ride the popularity
        if (_latestSentimentValue != null)
        {
            _targetHoldings = sentiment > _latestSentimentValue ? 1 : 0;
        }
        _latestSentimentValue = sentiment;
    }

    public override void OnData(Slice slice)
    {
        // Update trade direction based on updated data
        if (slice.ContainsKey(_datasetSymbol))
        {
            var sentiment = slice[_datasetSymbol].Sentiment;
            Update(sentiment);
        }

        // Ensure we have security data in the current slice to avoid stale fill
        // Buy if sentiment increase, liquidate otherwise to ride on the popularity of the equity
        if (slice.Bars.ContainsKey(_symbol) && _targetHoldings == 1 != Portfolio.Invested)
        {
            SetHoldings(_symbol, _targetHoldings);
        }
    }
}
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of US Equities that have been mentioned in an article over the last seven days. It then buys the subset of Equities that have increasing sentiment and forms an equal-weighted portfolio.

```C#
public class BrainSentimentDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 10);
        SetCash(100000);

        Settings.MinimumOrderMarginPortfolioPercentage = 0;
        UniverseSettings.Resolution = Resolution.Daily;
```

```
        UniverseSettings.Resolution = Resolution.Daily;
        // Filter by sentiment data
        AddUniverse<BrainSentimentIndicatorUniverse>(altCoarse =>
        {
            // Filter for any sentiment on last 7 days to trade on sentiment news
            return from d in altCoarse.OfType<BrainSentimentIndicatorUniverse>()
                where d.SentimentalArticleMentions7Days > 0m
                select d.Symbol;
        });

        AddAlpha(new BrainSentimentAlphaModel());

        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());

        AddRiskManagement(new NullRiskManagementModel());

        SetExecution(new ImmediateExecutionModel());
    }
}


public class BrainSentimentAlphaModel : AlphaModel
{
    private Dictionary<Symbol, SymbolData> _symbolDataBySymbol = new Dictionary<Symbol, SymbolData>();

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = new List<Insight>();

        foreach (var entry in _symbolDataBySymbol)
        {
            var symbol = entry.Key;
            var symbolData = entry.Value;

            // Update trade direction based on updated data
            if (slice.ContainsKey(symbolData.datasetSymbol)   && slice[symbolData.datasetSymbol] != null)
            {
                var sentiment = slice[symbolData.datasetSymbol].Sentiment;
                symbolData.Update(sentiment);
            }

            // Ensure we have security data in the current slice to avoid stale fill
            if (!(slice.ContainsKey(symbol)  && slice[symbol] != null))
            {
                continue;
            }

            // Buy if sentiment increase, liquidate otherwise to ride on the popularity of the equity
            if (symbolData.targetDirection == InsightDirection.Up  && !algorithm.Portfolio[symbol].Invested)
            {
                insights.Add(Insight.Price(symbol, TimeSpan.FromDays(100), symbolData.targetDirection));
            }
        }

        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            var symbol = security.Symbol;
            _symbolDataBySymbol.Add(symbol, new SymbolData(algorithm, symbol));
        }

        foreach (var security in changes.RemovedSecurities)
        {
            var symbol = security.Symbol;
            if (_symbolDataBySymbol.ContainsKey(symbol))
            {
                _symbolDataBySymbol[symbol].dispose();
                _symbolDataBySymbol.Remove(symbol);
            }
        }
    }
}


public class SymbolData
{
    public Symbol datasetSymbol;
    public InsightDirection targetDirection = InsightDirection.Flat;
```

```
    public InsightDirection targetDirection = InsightDirection.Flat;
    private QCAlgorithm _algorithm;
    private decimal? _latestSentimentValue = null;

    public SymbolData(QCAlgorithm algorithm, Symbol symbol)
    {
        _algorithm = algorithm;

        // Requesting the processed longer term (30-day) sentiment score data for sentiment trading
        datasetSymbol = algorithm.AddData<BrainSentimentIndicator30Day>(symbol).Symbol;

        // Historical data
        var history = algorithm.History<BrainSentimentIndicator30Day>(datasetSymbol, 100, Resolution.Daily);
        if (history.Count() == 0)
        {
            return;
        }

        // Warm up historical sentiment values, cache for comparing last sentiment score to trade, making it
immediately tradable signal
        var previousSentimentValues = history.Select(x => x.Sentiment);
        foreach (var sentiment in previousSentimentValues)
        {
            Update(sentiment);
        }
    }

    public void dispose()
    {
        // Unsubscribe from the Brain Sentiment feed for this security to release computational resources
        _algorithm.RemoveSecurity(datasetSymbol);
    }

    public void Update(decimal sentiment)
    {
        // Comparing the last sentiment score and decide to buy if the sentiment increases to ride the popularity
        if (_latestSentimentValue != null)
        {
            targetDirection = sentiment > _latestSentimentValue ? InsightDirection.Up : InsightDirection.Flat;
        }
        _latestSentimentValue = sentiment;
    }
}
```

Research Example

The following example lists US Equities having the highest 7-day sentiment.

```
C#
#r "../QuantConnect.DataSource.BrainSentiment.dll"
using QuantConnect.DataSource;

var qb = new QuantBook();

// Requesting data
var aapl = qb.AddEquity("AAPL", Resolution.Daily).Symbol;
var symbol = qb.AddData<BrainSentimentIndicator30Day>(aapl).Symbol;

// Historical data
var history = qb.History<BrainSentimentIndicator30Day>(symbol, 30, Resolution.Daily);
foreach (BrainSentimentIndicator30Day sentiment in history)
{
    Console.WriteLine($"{sentiment} at {sentiment.EndTime}");
}

// Add Universe Selection
IEnumerable<Symbol> UniverseSelection(IEnumerable<BaseData> altCoarse)
{
    return (from d in altCoarse.OfType<BrainSentimentIndicatorUniverse>()
        orderby d.Sentiment7Days descending select d.Symbol).Take(10);
}
var universe = qb.AddUniverse<BrainSentimentIndicatorUniverse>(UniverseSelection);

// Historical Universe data
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-5), qb.Time);
foreach (var sentiments in universeHistory)
{
    foreach (BrainSentimentIndicatorUniverse sentiment in sentiments)
    {
        Console.WriteLine($"{sentiment.Symbol} 7-day sentiment at {sentiment.EndTime}: {sentiment.Sentiment7Days}");
    }
}
```

Data Point Attributes

The Brain Sentiment Indicator dataset provides `BrainSentimentIndicatorBase` and `BrainSentimentIndicatorUniverse` objects.

BrainSentimentIndicatorBase Attributes

`BrainSentimentIndicatorBase` objects have the following attributes:

BrainSentimentIndicatorUniverse Attributes

`BrainSentimentIndicatorUniverse` objects have the following attributes:

Brain

Brain Wikipedia Page Views

Introduction

The Brain Wikipedia Page Views dataset by Brain tracks investor attention toward large U.S. equities by monitoring traffic on company Wikipedia pages. The dataset covers approximately 1,000 of the largest U.S. companies, broadly corresponding to the Russell 1000, and is available starting from February 2023. Data is delivered at a daily frequency.

The dataset is constructed by analyzing the number of views on each company's Wikipedia page, including both the main page and any secondary pages that redirect to it. Only English-language Wikipedia pages are considered. The data is published daily, typically by 12:00 PM UTC, and reflects attention metrics calculated relative to the reporting date.

In addition to raw page-view counts, the dataset includes buzz metrics, which normalize current attention levels relative to historical behavior. Buzz scores are computed as the distance between current page views and the company's own historical average over the previous six months, expressed in units of standard deviations. As a result, positive buzz values indicate unusually high attention, while negative values indicate lower-than-normal attention.

Metrics are provided across multiple time horizons, including 1-day, 7-day, and 30-day rolling windows, enabling users to study short-term spikes in attention as well as more persistent trends. These features make the dataset well-suited for research into investor attention, behavioral finance signals, event-driven strategies, and alpha models that incorporate information demand.

This dataset depends on the US Equity Security Master dataset, which provides essential information such as symbol mappings, corporate actions, and identifier consistency required for accurate historical analysis and backtesting.

For more information about the Brain Wikipedia Page Views dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Brain is a Research Company that creates proprietary datasets and algorithms for investment strategies, combining experience in financial markets with strong competencies in Statistics, Machine Learning, and Natural Language Processing. The founders share a common academic background of research in Physics as well as extensive experience in Financial markets.

Getting Started

The following snippet demonstrates how to request data from the Brain Wikipedia Page Views dataset:

```C#
_symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
_datasetSymbol = AddData<BrainWikipediaPageViews>(_symbol).Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	February 2023
Asset Coverage*	~1,000 US Equities (approximately Russell 1000)
Data Density	Sparse
Resolution	Daily
Timezone	UTC

* The coverage includes the largest U.S. equities since the start date and may evolve over time.

Requesting Data

To add Brain Wikipedia Page Views Indicator data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```
C#
public class BrainWikipediaPageViews: QCAlgorithm
{
    private Symbol _symbol;
    public override void Initialize()
    {
        SetStartDate(2021, 1, 1);
        SetEndDate(2021, 7, 8);
        SetCash(100000);
        _symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _datasetSymbol = AddData<BrainWikipediaPageViews>(_symbol).Symbol;
    }
}
```

Accessing Data

To get the current Brain Wikipedia Page Views Indicator data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var dataPoint = slice[_datasetSymbol];
        Log($"{_datasetSymbol} report sentiment at {slice.Time}: {dataPoint. NumberViews1}, {dataPoint. Buzz1}");
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<BrainWikipediaPageViews>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} metric at {slice.Time}: {data_point. NumberViews1}, {data_point. Buzz1}");
    }
}
```

Historical Data

To get historical Brain Wikipedia Page Views data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
var history = History<BrainWikipediaPageViews>(_datasetSymbol, 100, Resolution.Daily);
```

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_datasetSymbol);
```

Example Applications

The Brain Wikipedia Page Views dataset enables you to incorporate investor attention signals derived from Wikipedia traffic into your strategies. Examples include the following strategies:

- Buying securities that experience unusually high spikes in Wikipedia page views, indicating elevated public interest.
- Avoiding or reducing exposure to securities with declining or abnormally low attention relative to their historical baseline.
- Scaling position sizes based on short-term or long-term attention intensity using page view and buzz metrics.
- Building cross-sectional ranking strategies that favor stocks with sustained increases in public visibility.

Classic Algorithm Example

The following example algorithm ranks large-cap U.S. stocks daily using Brain Wikipedia Page Views attention and buzz metrics, then allocates portfolio weights proportionally to each stock's relative attention score.

C#

```
public class BrainWikipediaPageViewsRankingAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2025, 1, 1);
        SetEndDate(2025, 11, 23);
        SetCash(100000);

        // Seed prices so we don't get "no price" trading errors
        Settings.SeedInitialPrices = true;

        // We cherry picked 5 largest stocks, high trading volume provides better information and credibility for ML
ranking
        var tickers = new[] { "AAPL", "TSLA", "MSFT", "F", "KO" };

        foreach (var ticker in tickers)
        {
            // Requesting data to get 2 days estimated relative ranking
            var symbol = AddEquity(ticker, Resolution.Daily).Symbol;
            var datasetSymbol = AddData<BrainWikipediaPageViews>(symbol).Symbol;

            // Historical data (request dataset symbol)
            var history = History(datasetSymbol, 365, Resolution.Daily);
            Debug($"We got {history.Count()} items from our history request for {symbol}");
        }
    }

    public override void OnData(Slice slice)
    {
        // Get all BrainWikipediaPageViews points in the slice
        var points = slice.Get<BrainWikipediaPageViews>();
        if (points == null || points.Count == 0)
        {
            return;
        }

        var ranked = points.Values.ToDictionary(
            x => x.Symbol.Underlying,
            x =>
            {
                // Example composite signal:
                var number_of_views = x.NumberViews1.HasValue ? x.NumberViews1.Value : 0m;
                var buzz1day = x.Buzz1.HasValue ? x.Buzz1.Value : 0m;
                return buzz1day * (decimal)Math.Sqrt((double)number_of_views);
            })
            // Rank by score (higher score = better)
            .OrderByDescending(x => x.Value).ToList();

        // Convert ranks to weights (same style as your example)
        // best gets highest rank number
        var n = ranked.Count;
        var ranks = Enumerable.Range(1, n).Reverse().ToArray(); // n, n-1, ..., 1
        var denom = ranks.Sum();
        var top = string.Join(", ", ranked.Take(3).Select(x => $"{x.Key.Value}:{x.Value:0.0000}"));
        Debug($"{Time.Date:yyyy-MM-dd} top: {top}");

        // Place orders according to the ML ranking, the better the rank, the higher the estimated return and hence
weight
        for (var i = 0; i < ranked.Count; i++)
        {
            var w = (decimal)ranks[i] / denom;
            SetHoldings(ranked[i].Key, w);
        }
    }
}
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of liquid U.S. equities, ranks them daily using Brain Wikipedia Page Views attention and buzz metrics, and emits weighted bullish insights for the most attention-driven stocks using the Alpha Framework.

C#

```
public class BrainWikipediaPageViewsFrameworkRankingAlgorithm : QCAlgorithm
```

```
{
    public override void Initialize()
    {
        SetStartDate(2025, 1, 1);
        SetEndDate(2025, 11, 23);
        SetCash(100000);

        UniverseSettings.Resolution = Resolution.Daily;

        AddUniverse(f => f
            .Where(c => c.HasFundamentalData && c.Price > 10)
            .OrderByDescending(c => c.DollarVolume)
            .Take(50)
            .Select(c => c.Symbol));

        var alpha = new BrainWikipediaPageViewsRankingAlpha();
        AddAlpha(alpha);

        Schedule.On(
            DateRules.EveryDay(),
            TimeRules.At(9, 0),
            alpha.EmitInsights
        );

        SetPortfolioConstruction(new InsightWeightingPortfolioConstructionModel());
        AddRiskManagement(new NullRiskManagementModel());
        SetExecution(new ImmediateExecutionModel());

        SetWarmUp(30, Resolution.Daily);
    }
}

public class BrainWikipediaPageViewsRankingAlpha : AlphaModel
{
    private readonly Dictionary<Symbol, SymbolData> _symbolDataBySymbol = new();
    private readonly Dictionary<Symbol, Symbol> _symbolByDatasetSymbol = new();

    private readonly Dictionary<Symbol, double> _latestScores = new();
    private DateTime? _lastScoreDate;

    private QCAlgorithm _algorithm;

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        _algorithm = algorithm;

        var points = slice.Get<BrainWikipediaPageViews>();
        if (points == null)
        {
            return Enumerable.Empty<Insight>();
        }

        foreach (var kvp in points)
        {
            var dsSymbol = kvp.Key;
            var point = kvp.Value;

            if (!_symbolByDatasetSymbol.TryGetValue(dsSymbol, out var sym))
            {
                continue;
            }

            var numberOfViews = point.NumberViews1.HasValue ? (double)point.NumberViews1.Value : 0.0;
            var buzz1day = point.Buzz1.HasValue ? (double)point.Buzz1.Value : 0.0;

            var score = buzz1day + Math.Sqrt(numberOfViews);
            _latestScores[sym] = score;
        }

        _lastScoreDate = algorithm.Time.Date;

        return Enumerable.Empty<Insight>();
    }

    public void EmitInsights()
    {
        var algorithm = _algorithm;
        if (algorithm == null)
        {
            return;
        }
```

```
        }

        if (!_lastScoreDate.HasValue || _lastScoreDate.Value != algorithm.Time.Date)
        {
            return;
        }

        var paired = new List<(Symbol Symbol, double Score)>();
        foreach (var kvp in _latestScores)
        {
            var sym = kvp.Key;
            var sc = kvp.Value;

            if (!algorithm.Securities.TryGetValue(sym, out var sec) || sec == null || !sec.HasData || sec.Price <= 0)
            {
                continue;
            }

            if (sc == 0)
            {
                continue;
            }

            paired.Add((sym, sc));
        }

        if (paired.Count == 0)
        {
            return;
        }

        paired.Sort((a, b) => b.Score.CompareTo(a.Score));

        var n = paired.Count;
        var denom = n * (n + 1) / 2.0;

        var insights = new List<Insight>(n);
        for (int i = 0; i < n; i++)
        {
            var sym = paired[i].Symbol;
            var rank = n - i;
            var weight = rank / denom;

            insights.Add(Insight.Price(sym, TimeSpan.FromDays(7), InsightDirection.Up, null, null, null, weight));
        }

        algorithm.EmitInsights(insights.ToArray());

    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            var symbol = security.Symbol;
            if (_symbolDataBySymbol.ContainsKey(symbol))
            {
                continue;
            }

            var sd = new SymbolData(algorithm, symbol);
            _symbolDataBySymbol[symbol] = sd;
            _symbolByDatasetSymbol[sd.DatasetSymbol] = symbol;
        }

        foreach (var security in changes.RemovedSecurities)
        {
            var symbol = security.Symbol;

            if (_symbolDataBySymbol.TryGetValue(symbol, out var sd))
            {
                _symbolDataBySymbol.Remove(symbol);
                sd.Dispose();
            }

            Symbol toRemove = default;
            var found = false;
            foreach (var kvp in _symbolByDatasetSymbol)
            {
                if (kvp.Value == symbol)
                {
                    toRemove = kvp.Key;
```

```
                    found = true;
                    break;
                }
            }

            if (found)
            {
                _symbolByDatasetSymbol.Remove(toRemove);
            }

            _latestScores.Remove(symbol);
        }
    }
}

public class SymbolData
{
    private readonly QCAlgorithm _algorithm;
    public Symbol DatasetSymbol { get; }

    public SymbolData(QCAlgorithm algorithm, Symbol symbol)
    {
        _algorithm = algorithm;

        DatasetSymbol = algorithm.AddData<BrainWikipediaPageViews>(symbol, Resolution.Daily).Symbol;

        var history = algorithm.History(DatasetSymbol, 365, Resolution.Daily);
        algorithm.Debug($"We got {history.Count()} items from our history request for {symbol}");
    }

    public void Dispose()
    {
        _algorithm.RemoveSecurity(DatasetSymbol);
    }
}
```

Research Example

The following example retrieves historical Brain Wikipedia Page Views data for Apple and prints the daily number of Wikipedia page views over the past six months.

C#

```
#r "../QuantConnect.DataSource.BrainSentiment.dll"
var qb = new QuantBook();
// Requesting data
var aapl = qb.AddEquity("AAPL", Resolution.Daily).Symbol;
var symbol = qb.AddData<BrainWikipediaPageViews>(aapl).Symbol;

// Historical data
var history = qb.History<BrainWikipediaPageViews>(symbol, 180, Resolution.Daily);
foreach (BrainWikipediaPageViews PageMetrics in history)
{
    Console.WriteLine($"{PageMetrics} at {PageMetrics.EndTime}");
}
```

Data Point Attributes

The Brain Wikipedia Page Views dataset provides `BrainWikipediaPageViews` objects.

`BrainWikipediaPageViews` objects have the following attributes:

Datasets

CoinGecko

CoinGecko was founded in 2014 by TM Lee (CEO) and Bobby Ong (COO) with the mission to democratize the access of crypto data and empower users with actionable insights. We also deep dive into the crypto space to deliver valuable insights to our users through our cryptocurrency reports, as well as our publications, newsletter, and more.

Crypto Market Cap

CoinGecko

Crypto Market Cap

Introduction

The Crypto Market Cap dataset by CoinGecko tracks the market cap of cryptocurrencies. The data covers 620 cryptocurrencies that are supported by QuantConnect, starts in 28 April 2013, and is delivered on a daily frequency. This dataset is created by scraping CoinGecko's Market Chart.

For more information about the Crypto Market Cap dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

CoinGecko was founded in 2014 by TM Lee (CEO) and Bobby Ong (COO) with the mission to democratize the access of crypto data and empower users with actionable insights. We also deep dive into the crypto space to deliver valuable insights to our users through our cryptocurrency reports, as well as our publications, newsletter, and more.

Getting Started

The following snippet demonstrates how to request data from the CoinGecko Crypto Market Cap dataset:

```C#
_symbol = AddData<CoinGecko>("BTC").Symbol;
_universe = AddUniverse(CoinGeckoUniverse, UniverseSelection)
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	28 April 2013
Asset Coverage	620 cryptocurrencies
Resolution	Daily
Timezone	UTC

Requesting Data

To add CoinGecko Crypto Market Cap data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```
C#
public class CoinGeckoMarketCapDataAlgorithm: QCAlgorithm
{
    private Symbol _symbol, _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);

        _symbol = AddCrypto("BTCUSD", Resolution.Daily).Symbol;
        _datasetSymbol = AddData<CoinGecko>("BTC").Symbol;
    }
}
```

Accessing Data

To get the current Crypto Market Cap data, index the current `Slice` with the dataset `Symbol` . `Slice` objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var dataPoint = slice[_datasetSymbol];
        Log($"{_datasetSymbol} market cap::volume at {slice.Time}: {dataPoint.MarketCap}::{dataPoint.Volume}");
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<CoinGecko>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} market cap::volume at {slice.Time}: {dataPoint.MarketCap}::{dataPoint.Volume}");
    }
}
```

Historical Data

To get historical CoinGecko Crypto Market Cap data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
var history = History<CoinGecko>(_datasetSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of Cryptos based on CoinGecko Crypto Market Cap data, call the `AddUniverse` method with the `CoinGeckoUniverse` class and a selection function. Note that the filtered output is a list of names of the coins. If corresponding tradable crypto pairs are preferred, call `CreateSymbol(market, quoteCurrency)` method for each output item.

```C#
public class CryptoMarketCapUniverseAlgorithm : QCAlgorithm
{
    private Universe _universe;
    public override void Initialize()
    {
        SetAccountCurrency("USD");
        var market = Market.Coinbase;
        var marketPairs = SymbolPropertiesDatabase.GetSymbolPropertiesList(market)
            .Where(x => x.Value.QuoteCurrency == AccountCurrency)
            .Select(x => x.Key.Symbol)
            .ToList();
        _universe = AddUniverse<CoinGecko>(data =>
        {
            foreach (var datum in data.OfType<CoinGecko>())
            {
                Debug($"{datum.Coin},{datum.MarketCap},{datum.Price}");
            }
            return data
                .Select(c => c as CoinGecko)
                // Select the coins that the brokerage supports and have a quote currency that
                // matches our account currency.
                .Where(c => marketPairs.Contains(c.Coin + AccountCurrency))
                // Select the 10 largest coins.
                .OrderByDescending(c => c.MarketCap)
                .Take(10)
                // Create the Symbol objects of the selected coins.
                .Select(c => c.CreateSymbol(market, AccountCurrency));
        });
    }
}
```

For more information about dynamic universes, see Universes .

Universe History

You can get historical universe data in an algorithm and in the Research Environment.

Historical Universe Data in Algorithms

To get historical universe data in an algorithm, call the `History` method with the `Universe` object and the lookback period. If there is no data in the period you request, the history result is empty.

```C#
var universeHistory = History(_universe, 30, Resolution.Daily);
foreach (var coins in universeHistory)
{
    foreach (CoinGecko coin in coins)
    {
        Log($"{coin.Symbol.Value} market cap at {coin.EndTime}: {coin.MarketCap}");
    }
}
```

Historical Universe Data in Research

To get historical universe data in research, call the `UniverseHistory` method with the `Universe` object, a start date, and an end date. This method returns the filtered universe. If there is no data in the period you request, the history result is empty.

```C#
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var coins in universeHistory)
{
    foreach (CoinGecko coin in coins)
    {
        Console.WriteLine($"{coin.Symbol.Value} market cap at {coin.EndTime}: {coin.MarketCap}");
    }
}
```

You can call the `History` method in Research.

Remove Subscriptions

To remove your subscription to CoinGecko Crypto Market Cap data, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_datasetSymbol);
```

Supported Assets

The following table shows the available Cryptocurrencies:

Supported Cryptocurrency					
1INCH	AAA	AAPL	AAVE	ACA	ACH
ACM	ADA	ADX	AE	AERGO	AGIX
AGLD	AION	AKRO	ALBT	ALCX	ALEPH
ALGO	ALICE	ALPACA	ALPHA	ALPINE	AMB
AMC	AMP	AMPL	ANC	ANKR	ANT
ANY	APE	API3	APPC	APT	AR
ARDR	ARK	ARPA	ASD	ASM	ASR
AST	ASTR	ATA	ATLAS	ATM	ATOM
AUCTION	AUDIO	AURY	AUTO	AVA	AVAX
AVT	AXS	B21	BADGER	BAKE	BAL
BAND	BAO	BAR	BAT	BB	BCC
BCD	BCH	BCN	BCPT	BDOT	BEAM
BEL	BEST	BETA	BETH	BFT	BICO
BIDR	BIFI	BIT	BITO	BLT	BLZ
BMI	BNB	BNT	BNTX	BNX	BOBA
BOND	BOSON	BOT	BRD	BRL	BRZ
BSV	BSW	BTC	BTCB	BTCST	BTG
BTRST	BTS	BTSE	BTT	BULL	BURGER
BUSD	BVOL	BYND	BZRX	C98	CAKE
CCD	CDT	CEL	CELO	CELR	CFX
CGC	CHAT	CHESS	CHEX	CHR	CHSB

CHZ	CITY	CKB	CLO	CLOAK	CLV
CMT	CND	CNHT	COCOS	COIN	COMP
CONV	COPE	COS	COTI	COVAL	COVER
CQT	CREAM	CRO	CRON	CRV	CTK
CTSI	CTX	CTXC	CUSDT	CVC	CVP
CVX	DAI	DAR	DASH	DATA	DAWN
DCR	DDX	DEGO	DENT	DESO	DEXE
DF	DFL	DGB	DGD	DIA	DLT
DMG	DNT	DOCK	DODO	DOGE	DORA
DOT	DREP	DUSK	DVF	DYDX	EDEN
EGLD	ELF	EMB	ENG	ENJ	ENS
EOS	EPS	EPX	ERN	ESS	ETC
ETH	ETH2	ETHE	ETP	EURS	EURT
EVX	EWT	EXO	EZ	FARM	FB
FCL	FET	FIDA	FIL	FIO	FIRO
FIS	FLM	FLOW	FLUX	FOR	FORTH
FOX	FRONT	FTM	FTT	FUEL	FUN
FX	FXS	GAL	GALA	GAS	GENE
GFI	GHST	GLD	GLM	GLMR	GLXY
GME	GMT	GMX	GNO	GNT	GO
GODS	GOG	GOT	GRS	GRT	GT
GTC	GTO	GTX	GVT	GYEN	HARD
HBAR	HC	HEGIC	HFT	HGET	HIGH
HIVE	HMT	HNT	HOLY	HOOK	HOT
HT	HUM	HXRO	ICE	ICN	ICP
ICX	ID	IDEX	IDRT	ILV	IMX
INJ	INS	INTER	INV	IOST	IOTX
IQ	IRIS	JASMY	JET	JOE	JST

JUV	KAI	KAN	KAR	KAVA	KDA
KEEP	KEY	KIN	KLAY	KMD	KNC
KP3R	KRL	KSHIB	KSM	LAZIO	LCX
LDO	LEND	LEO	LEVER	LINA	LINK
LIT	LOKA	LOOKS	LOOM	LPT	LQTY
LRC	LSK	LTC	LTO	LUA	LUN
LUNA	LUNC	LYM	MAGIC	MANA	MAPS
MASK	MATH	MATIC	MBL	MBOX	MBS
MC	MCB	MCO	MCO2	MDA	MDT
MDX	MEDIA	MER	MFT	MIM	MINA
MIR	MITH	MKR	MLN	MNGO	MOB
MOD	MOVR	MPL	MSOL	MSTR	MTA
MTH	MTL	MULTI	MUSD	NAS	NAV
NBS	NCASH	NCT	NEAR	NEBL	NEO
NEXO	NKN	NMR	NPXS	NU	NULS
NXS	OAX	OCEAN	ODE	OG	OGN
OKB	OM	OMG	OMNI	ONE	ONG
ONT	OOKI	OP	ORBS	ORCA	ORN
ORS	OSMO	OST	OXT	OXY	PASS
PAXG	PEOPLE	PERL	PERP	PHA	PHB
PHX	PIVX	PLA	PLANETS	PLU	PNG
PNK	PNT	POA	POE	POLIS	POLS
POLY	POLYX	POND	PORT	PORTO	POWR
PPT	PRISM	PRO	PROM	PROS	PSG
PTU	PUNDIX	PYR	QI	QKC	QLC
QNT	QRDO	QSP	QTF	QTUM	QUICK
RAD	RAI	RAMP	RARE	RARI	RAY
RBN	RBTC	RCN	RDN	REAL	REEF

REI	REN	RENBTC	REP	REQ	RGT
RIF	RLC	RLY	RNDR	RON	ROOK
ROSE	RRT	RSR	RUNE	RVN	SALT
SAN	SAND	SANTOS	SC	SCRT	SFP
SGB	SHIB	SHPING	SKL	SKY	SLND
SLP	SLRS	SNGLS	SNM	SNT	SNX
SNY	SOL	SOS	SPARTA	SPELL	SPY
SRM	SSV	STARS	STEEM	STEP	STETH
STG	STMX	STORJ	STORM	STPT	STRAX
STSOL	STX	SUB	SUKU	SUN	SUPER
SUSD	SUSHI	SWRV	SXP	SYS	T
TBTC	TCT	TFUEL	THETA	TKO	TLM
TLOS	TNB	TNT	TOMO	TORN	TRAC
TRADE	TRB	TRIBE	TROY	TRU	TRX
TRYB	TSLA	TULIP	TUSD	TVK	TWT
UFT	UMA	UNFI	UNI	UOS	UPI
USDC	USDP	USDS	USDT	USTC	UTK
VAI	VEE	VELO	VEN	VET	VGX
VIA	VIB	VIBE	VIDT	VITE	VOXEL
VTHO	WABI	WAN	WAVES	WAXP	WBTC
WCFG	WILD	WIN	WING	WINGS	WNCG
WNDR	WNXM	WOO	WPR	WRX	WTC
XAUT	XCHF	XDC	XDG	XEC	XEM
XLM	XMR	XNO	XRA	XRD	XRP
XTZ	XVG	XVS	XYO	YFI	YFII
YGG	ZCN	ZEC	ZEN	ZIL	ZM
ZMT	ZRX				

Example Applications

The CoinGecko Crypto Market Cap dataset provides information on the size of the crypto coin and can be used to compare the size of one coin to another. Examples include the following strategies:

- Construct a major crypto index fund.
- Invest in the cryptos with the fastest growth in market size.
- Red flag stop when there might be a crypto bank run.

Classic Algorithm Example

The following example algorithm buy BTCUSD when the market cap of BTC is rising, while sell it when the market cap of BTC is dropping.

```C#
public class CoinGeckoAlgorithm : QCAlgorithm
{
    private Symbol _cryptoSymbol, _customDataSymbol;
    private RollingWindow<CoinGecko> _window;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request BTCUSD crypto data for trading
        _cryptoSymbol = AddCrypto("BTCUSD").Symbol;
        // Request CoinGecko Market Cap data of BTC for trade signal generation
        _customDataSymbol = AddData<CoinGecko>("BTC").Symbol;
        // Use RollingWindow to save the last 2 market cap data for capital flow analysis
        _window = new RollingWindow<CoinGecko>(2);
    }

    public override void OnData(Slice slice)
    {
        // Trade based on updated market cap data
        var data = slice.Get<CoinGecko>();
        if (!data.IsNullOrEmpty() && data.ContainsKey(_customDataSymbol))
        {
            // Update RollingWindow for updated comparison
            _window.Add(data[_customDataSymbol]);
            if (!_window.IsReady)
            {
                return;
            }

            // Buy BTCUSD if the market cap of BTC is increasing, which suggests the capital flow towards BTC market
and drive up the demand
            if (_window[0].MarketCap > _window[1].MarketCap)
            {
                SetHoldings(_cryptoSymbol, 1);
            }
            // Sell otherwise, since the capital is flowing out and the demand of BTC lowered
            else
            {
                SetHoldings(_cryptoSymbol, -0.5m);
            }
        }
    }

    public override void OnOrderEvent(OrderEvent orderEvent)
    {
        if (orderEvent.Status.IsFill())
        {
            Debug($"Purchased Crypto: {orderEvent.Symbol}");
        }
    }
}
```

Framework Algorithm Example

The following example algorithm buy BTCUSD when the market cap of BTC is rising, while sell it when the market cap of BTC is

dropping with framework implementation.

```C#
public class CoinGeckoAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request CoinGecko Market Cap data of BTC for trade signal generation
        var symbol = AddData<CoinGecko>("BTC").Symbol;
        // Request BTCUSD crypto data for trading
        var crypto = AddCrypto("BTCUSD", market: Market.Coinbase).Symbol;
        var symbolDict = new Dictionary<Symbol, Symbol> { {symbol, crypto} };
        // Use RollingWindow to save the last 2 market cap data for capital flow analysis
        var window = new Dictionary<Symbol, RollingWindow<CoinGecko>>{ {symbol, new RollingWindow<CoinGecko>(2)} };

        // Custom alpha model that emit insights based on updated market cap data
        AddAlpha(new CoinGeckoAlphaModel(symbolDict, window));
        // Equal invest to evenly dissipate capital concentration risk from non-systematic individual risky event
        SetPortfolioConstruction(new InsightWeightingPortfolioConstructionModel());
    }
}

public class CoinGeckoAlphaModel : AlphaModel
{
    private Dictionary<Symbol, Symbol> _symbolDict = new();
    private Dictionary<Symbol, RollingWindow<CoinGecko>> _window = new();

    public CoinGeckoAlphaModel(Dictionary<Symbol, Symbol> symbolDict, Dictionary<Symbol, RollingWindow<CoinGecko>>
window)
    {
        _symbolDict = symbolDict;
        _window = window;
    }

    public override List<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = new List<Insight>();

        // Trade based on updated market cap data
        var data = slice.Get<CoinGecko>();

        foreach (var kvp in _symbolDict)
        {
            var datasetSymbol = kvp.Key;
            var cryptoSymbol = kvp.Value;

            if (!data.ContainsKey(kvp.Key)) continue;
            // Update RollingWindow for updated comparison
            _window[datasetSymbol].Add(data[datasetSymbol]);

            // Can only compare market cap if the RollingWindow is ready
            var window = _window[datasetSymbol];
            if (!window.IsReady) continue;

            // Buy BTCUSD if the market cap of BTC is increasing, which suggests the capital flow towards BTC market
and drive up the demand
            if (window[0].MarketCap > window[1].MarketCap)
            {
                var insight = Insight.Price(cryptoSymbol, TimeSpan.FromDays(1), InsightDirection.Up, weight: 1d);
                insights.Add(insight);
            }
            // Sell otherwise, since the capital is flowing out and the demand of BTC lowered
            else
            {
                var insight = Insight.Price(cryptoSymbol, TimeSpan.FromDays(1), InsightDirection.Down, weight:
0.25d);
                insights.Add(insight);
            }
        }

        return insights;
    }
}
```

Research Example

The following example lists US Equities having the highest 7-day sentiment.

```C#
#r "../QuantConnect.DataSource.CoinGecko.dll"
using QuantConnect.DataSource;

var qb = new QuantBook();

// Requesting Data
var symbol = qb.AddData<CoinGecko>("BTC").Symbol;

// Historical data
var history = qb.History<CoinGecko>(symbol, 30, Resolution.Daily);
foreach (CoinGecko coin in history)
{
    Console.WriteLine($"{coin} at {coin.EndTime}");
}

// Add Universe Selection
IEnumerable<Symbol> UniverseSelection(IEnumerable<BaseData> altCoarse)
{
    return (from d in altCoarse.OfType<CoinGecko>()
        orderby d.MarketCap descending select d.Symbol).Take(10);
}
var universe = qb.AddUniverse<CoinGeckoUniverse<(UniverseSelection);

// Historical Universe data
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-5), qb.Time);
foreach (var coins in universeHistory)
{
    foreach (CoinGecko coin in coins)
    {
        Console.WriteLine($"{coin.Symbol.Value} market cap at {coin.EndTime}: {coin.MarketCap}");
    }
}
```

Data Point Attributes

The Crypto Market Cap dataset provides `CoinGecko` and `CoinGeckoUniverse` objects.

CoinGecko Attributes

`CoinGecko` objects have the following attributes:

CoinGeckoUniverse Attributes

`CoinGeckoUniverse` objects have the following attributes:

Datasets

EOD Historical Data

EODHD was a France financial data provider founded in April 2015. They focus on providing clean financial data, including stock prices, splits, dividends, fundamentals, macroeconomy indicators, technical indicators, and alternative data sources, through 24/7 API seamlessly.

Economic Events

Macroeconomics Indicators

Upcoming Dividends

Upcoming Earnings

Upcoming IPOs

Upcoming Splits

EOD Historical Data

Economic Events

Introduction

The Economic Events dataset, provided by EODHD, offers daily alerts for major economic events or announcements of global markets within the upcoming 7 days, with estimation and previous record if available. The data starts in January 2019, and is delivered on a daily frequency.

For more information about the Economic Events dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

EODHD was a France financial data provider founded in April 2015. They focus on providing clean financial data, including stock prices, splits, dividends, fundamentals, macroeconomy indicators, technical indicators, and alternative data sources, through 24/7 API seamlessly.

Getting Started

The following snippet demonstrates how to request data from the Economic Events dataset:

```C#
AddData<EODHDEconomicEvents>(Country.UnitedStates);
AddData<EODHDEconomicEvents>(EODHD.Events.UnitedStates.Cpi);
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2019
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Requesting Data

To add Economic Events data to your algorithm, call the `AddData<EODHDUpcomingIPOs>` method.

```
C#
public class EconomicEventsDataAlgorithm : QCAlgorithm
{
    private Symbol _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);

        var ticker = EODHD.Events.UnitedStates.Cpi;
        // var ticker = Country.UnitedStates;
        _datasetSymbol = AddData<EODHDEconomicEvents>(ticker).Symbol;
    }
}
```

Accessing Data

To get the current Economic Events data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.Get<EODHDEconomicEvents>().TryGetValue(_datasetSymbol, out var economicEvents))
    {
        foreach (EODHDEconomicEvent economicEvent in economicEvents)
        {
           Log($"{economicEvent.EventType} of {economicEvent.Country} over {economicEvent.EventPeriod} with be held
at {economicEvent.EventTime} with estimated value {economicEvent.Estimate}");
        }
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    foreach (var (symbol, economicEvents) in slice.Get<EODHDEconomicEvents>())
    {
        foreach (EODHDEconomicEvent economicEvent in economicEvents)
        {
           Log($"{economicEvent.EventType} of {economicEvent.Country} over {economicEvent.EventPeriod} with be held
at {economicEvent.EventTime} with estimated value {economicEvent.Estimate}");
        }
    }
}
```

Historical Data

To get historical Economic Events data, call the `History` method with the type `EODHDEconomicEvents` cast and the underlying `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
var history = History<EODHDEconomicEvents>(_datasetSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```
RemoveSecurity(_datasetSymbol);
```

Example Applications

The Economic Events dataset provides timely notifications about upcoming economic events, allowing traders to trade on that volatility. Examples include the following strategies:

- Long straddle to trade the volatility of the economic events.
- Trade market representative ETFs based on estimated figures for the upcoming events.
- Statistical arbitrage on two or more correlated global markets based on economic events of different locations.

Classic Algorithm Example

The following example algorithm trade Industrial sector ETF based on US PMI estimated change direction. If the estimated PMI of the upcoming announcement is above the previous PMI, buy the Industrial ETF, and sell it otherwise.

```
public class EODHDEconomicEventsAlgorithm : QCAlgorithm
{
    private Symbol _equitySymbol, _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Use industrial sector ETF as a vehicle to trade.
        _equitySymbol = AddEquity("XLI").Symbol;
        // Request US PMI economic event data to generate trade signals.
        var ticker = EODHD.Events.UnitedStates.MarkitManufacturingPurchasingManagersIndex;
        _datasetSymbol = AddData<EODHDEconomicEvents>(ticker).Symbol;
    }

    public override void OnData(Slice slice)
    {
        // Trade based on the updated economic events.
        if (slice.Get<EODHDEconomicEvents>().TryGetValue(_datasetSymbol, out var economicEvents))
        {
            var economicEvent = economicEvents.FirstOrDefault() as EODHDEconomicEvent;
            // Use the Manufacturing Index to generate trade signals on manufacturing industry vehicles.
            // Make sure previous and estimate are available to estimate the direction of the industry.
            if (economicEvent.Previous.HasValue && economicEvent.Estimate.HasValue)
            {
                //If the estimated PMI is higher than the previous PMI, the manufacturing ETF price is expected to
rise.
                if (economicEvent.Previous.Value > economicEvent.Estimate.Value)
                {
                    SetHoldings(_equitySymbol, 1);
                }
                // Otherwise, it is expected manufacturing ETF price will drop.
                else
                {
                    SetHoldings(_equitySymbol, -1);
                }
            }
        }
    }
}
```

Framework Algorithm Example

The following example algorithm trade Industrial sector ETF based on US PMI estimated change direction using algorithm framework. If the estimated PMI of the upcoming announcement is above the previous PMI, buy the Industrial ETF, sell it otherwise.

```C#
public class EODHDEconomicEventsAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Use industrial sector ETF as a vehicle to trade.
        var symbol = QuantConnect.Symbol.Create("XLI", SecurityType.Equity, Market.USA);
        AddUniverseSelection(new ManualUniverseSelectionModel(new[] {symbol}));
        // Custom alpha model to emit insights according to economic events.
        AddAlpha(new EconomicEventAlphaModel(this));
        // Equal weighting position sizing to dissipate capital risk evenly.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel(Expiry.EndOfMonth));
    }
}

public class EconomicEventAlphaModel : AlphaModel
{
    private Symbol _symbol, _datasetSymbol;

    public EconomicEventAlphaModel(QCAlgorithm algorithm)
    {
        // Request US PMI economic event data to generate trade signals.
        var ticker = EODHD.Events.UnitedStates.MarkitManufacturingPurchasingManagersIndex;
        _datasetSymbol = algorithm.AddData<EODHDEconomicEvents>(ticker).Symbol;
    }

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        // Trade based on the updated economic events.
        if (slice.Get<EODHDEconomicEvents>().TryGetValue(_datasetSymbol, out var economicEvents))
        {
            // Use the US Manufacturing Index to generate trade signals on manufacturing industry vehicles.
            // Make sure previous and estimate are available to estimate the direction of the industry.
            var economicEvent = economicEvents.FirstOrDefault() as EODHDEconomicEvent;
            if (economicEvent.Previous.HasValue && economicEvent.Estimate.HasValue)
            {
                // If the estimated PMI is higher than the previous PMI, the manufacturing ETF price is expected to
rise.
                // Otherwise, it is expected manufacturing ETF prices will drop.
                var direction = economicEvent.Previous.Value > economicEvent.Estimate.Value
                    ? InsightDirection.Up
                    : InsightDirection.Down;
                yield return Insight.Price(_symbol, TimeSpan.FromDays(5), direction);
            }
        }
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach (var added in changes.AddedSecurities)
        {
            _symbol = added.Symbol;
        }
    }
}
```

Data Point Attributes

The EODHD Economic Events dataset provides `EODHDEconomicEvent` objects, which have the following attributes:

The `EODHDEconomicEvents` objects represent a collection of `EODHDEconomicEvent` object

Country Enumeration

To filter the countries of the economic events, you can make use of the `Country` enumeration. The `Country` enumeration has the following members:

Events Enumeration

To filter the event types of the economic events, you can make use of the `Events` enumeration. The `Events` enumeration has the

following members:

EOD Historical Data

Macroeconomics Indicators

Introduction

The Macroeconomic Indicators dataset, provided by EODHD, provides data on 39 major macroeconomic indicators of 249 countries and regions. The data starts in January 1998, and is delivered on indicators update.

For more information about the Macroeconomics Indicators dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

EODHD was a France financial data provider founded in April 2015. They focus on providing clean financial data, including stock prices, splits, dividends, fundamentals, macroeconomy indicators, technical indicators, and alternative data sources, through 24/7 API seamlessly.

Getting Started

The following snippet demonstrates how to request data from the Macroeconomic Indicators dataset:

```C#
AddData<EODHDMacroIndicators>(Country.UnitedStates);
AddData<EODHDMacroIndicators>(EODHD.MacroIndicators.UnitedStates.GdpCurrentUsd);
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 1998
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Requesting Data

To add Macroeconomic Indicators data to your algorithm, call the `AddData<EODHDMacroIndicators>` method.

```C#
public class EODHDMacroIndicatorsAlgorithm : QCAlgorithm
{
    private Symbol _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);

        _datasetSymbol = AddData<EODHDMacroIndicators>(
            EODHD.MacroIndicators.UnitedStates.GdpCurrentUsd
        ).Symbol;
        // _datasetSymbol = AddData<EODHDMacroIndicators>(Country.UnitedStates).Symbol;
    }
}
```

Accessing Data

To get the current Macroeconomic Indicators data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var dataPoint = slice[_datasetSymbol];
        Log($"{dataPoint.Frequency} {dataPoint.Indicator} of {dataPoint.Country} at {dataPoint.Time} with value
{dataPoint.Value}");
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<EODHDMacroIndicators>())
    {
        var eventDataPoint = kvp.Value;
        Log($"{dataPoint.Frequency} {dataPoint.Indicator} of {dataPoint.Country} at {dataPoint.Time} with value
{dataPoint.Value}");
    }
}
```

Historical Data

To get historical Macroeconomic Indicators data, call the `History` method with the type `EODHDMacroIndicators` cast and the underlying `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
var history = History<EODHDMacroIndicators>(_datasetSymbol, 100);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```
RemoveSecurity(_datasetSymbol);
```

Example Applications

The Macroeconomic Indicators dataset provides data on global macro indicators, allowing traders to react to market sentiment. Examples include the following strategies:

- Long straddle to trade the market shock brought by the macro indicators.
- Hold long or short position of the market representative ETF based on the expected market direction.
- Statistical arbitrage on 2 or more correlated global markets based on market strength indicated by macro indicators.

Classic Algorithm Example

The following example algorithm holds long position of SPY if the US GDP is in positive growth, while holds short position if it is the opposite case.

```
public class EODHDMacroIndicatorsAlgorithm : QCAlgorithm
{
    private Symbol _equitySymbol, _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        _equitySymbol = AddEquity("SPY", Resolution.Daily).Symbol;
        var ticker = EODHD.MacroIndicators.UnitedStates.GdpGrowthAnnual;
        _datasetSymbol = AddData<EODHDMacroIndicators>(ticker).Symbol;
    }

    public override void OnData(Slice slice)
    {
        if (slice.Get<EODHDMacroIndicators>().TryGetValue(_datasetSymbol, out var indicators))
        {
            var gdp = indicators.FirstOrDefault() as EODHDMacroIndicator;
            SetHoldings(_equitySymbol, gdp.Value > 0m ? 1 : -1);
        }
    }
}
```

Framework Algorithm Example

The following example algorithm holds the long position of SPY if the US GDP is in positive growth while holding a short position if it is the opposite case. It is implemented in the framework algorithm.

```C#
public class EODHDMacroIndicatorsAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Use market ETF as a vehicle to trade.
        var symbol = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
        AddUniverseSelection(new ManualUniverseSelectionModel(symbol));
        // Custom alpha model to emit insights according to macroeconomic indicators.
        AddAlpha(new MacroIndicatorAlphaModel(this));
        // Equal weighting position sizing to dissipate capital risk evenly.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel(Expiry.EndOfMonth));
    }
}

public class MacroIndicatorAlphaModel : AlphaModel
{
    private Symbol _symbol, _datasetSymbol;

    public MacroIndicatorAlphaModel(QCAlgorithm algorithm)
    {
        // Request US GDP growth indicator data to generate trade signals.
        var ticker = EODHD.MacroIndicators.UnitedStates.GdpGrowthAnnual;
        _datasetSymbol = algorithm.AddData<EODHDMacroIndicators>(ticker).Symbol;
    }

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        // Trade based on the updated macroeconomic indicator.
        if (slice.Get<EODHDMacroIndicators>().TryGetValue(_datasetSymbol, out var indicators))
        {
            var gdp = indicators.FirstOrDefault() as EODHDMacroIndicator;
            // If the GDP growth is positive, the market ETF is expected to rise.
            // Otherwise, it is expected the market ETF will drop.
            var direction = gdp.Value > 0m ? InsightDirection.Up : InsightDirection.Down;
            yield return Insight.Price(_symbol, TimeSpan.FromDays(5), direction);
        }
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach (var added in changes.AddedSecurities)
        {
            _symbol = added.Symbol;
        }
    }
}
```

Data Point Attributes

The EODHD Macro Indicators dataset provides `EODHDMacroIndicators` objects, which have the following attributes:

MacroIndicators Enumeration

To filter the specific indicator, you can make use of the `MacroIndicators` enumeration. The `MacroIndicators` enumeration has the following members:

Country Enumeration

To filter the country of the macro indicators, you can make use of the `Country` enumeration. The `Country` enumeration has the following members:

Frequency Enumeration

To filter the indicator frequency of the macro indicators, you can make use of the `Frequency` enumeration. The `Frequency` enumeration has the following members:

EOD Historical Data

Upcoming Dividends

Introduction

The Upcoming Dividends dataset, provided by EODHD, offers daily alerts for US Equities that will have a dividend event within the upcoming 7 days. The data starts in January 2015 and is delivered on a daily frequency.

Compared to US Equity Security Master as a benchmark, the Upcoming Dividends dataset has a 98.56% coverage of all dividend events, while having a 99.71% precision on the exact dividend dates of the covered ones and a 99.90% precision within +/- 3 days.

For more information about the Upcoming Dividends dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

EOD Historical Data (EODHD) is a financial data provider based in France, and founded in April 2015. They focus on providing clean financial data, including stock prices, splits, dividends, fundamentals, macroeconomic indicators, technical indicators, and alternative data sources, through 24/7 API seamlessly. For more information about EODHD, visit https://eodhd.com/ .

Getting Started

The following snippet demonstrates how to request data from the Upcoming Dividends dataset:

```C#
using QuantConnect.DataSource;
AddUniverse<EODHDUpcomingDividends>(SelectionFunction);
AddData<EODHDUpcomingDividends>("dividends");
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2015
Data Density	Sparse
Resolution	Daily
Timezone	New York

Universe Selection

To select a dynamic universe of US Equities based on the Upcoming Dividends dataset, call the `AddUniverse` method with a `EODHDUpcomingDividends` cast.

```
C#
public override void Initialize()
{
    _universe = AddUniverse<EODHDUpcomingDividends>(UniverseSelectionFilter);
}

private IEnumerable<Symol> UniverseSelectionFilter(IEnumerable<BaseData> dividends)
{
    return from EODHDUpcomingDividends d in dividends
           where d.DividendsDate <= Time.AddDays(1) && d.Dividends > 0.05m
           select d.Symbol;
}
```

For more information about universe settings, see Settings .

Requesting Data

To add Upcoming Dividends data to your algorithm, call the `AddData<EODHDUpcomingDividends>` method.

```
C#
public class UpcomingDividendsDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);
        _symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _datasetSymbol = AddData<EODHDUpcomingDividends>("dividends").Symbol;
    }
}
```

Accessing Data

To get the current Upcoming Dividends data, call the `Get<EODHDUpcomingDividends>` method from the current `Slice` and index the result with the security `Symbol` . `Slice` objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    var upcomingDividends = slice.Get<EODHDUpcomingDividends>();
    if (upcomingDividends.TryGetValue(_symbol, out var upcomingDividendsDataPoint))
    {
        Log($"{_symbol} will pay dividend at {upcomingDividendsDataPoint.DividendDate} with dividend per share of
${upcomingDividendsDataPoint.Dividend}");
    }
}
```

You can also iterate through all of the dataset objects in the current `Slice`

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<EODHDUpcomingDividends>())
    {
        var equitySymbol = kvp.Key;
        var upcomingDividendsDataPoint = kvp.Value;
        Log($"{equitySymbol} will pay dividend at {upcomingDividendsDataPoint.DividendDate} with dividend per share
of ${upcomingDividendsDataPoint.Dividend}");
    }
}
```

Historical Data

To get historical Upcoming Dividends data, call the `History` method with the type `EODHDUpcomingDividends` cast and the period of request. If there is no data in the period you request, the history result is empty.

```C#
var history = History<EODHDUpcomingDividends>(TimeSpan.FromDays(100), Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_datasetSymbol);
```

Example Applications

The Upcoming Dividends dataset allows traders to trade the price change due to dividends. Examples include the following strategies:

- Short the stocks on dividend report day to earn the dividend discount on pricing.
- Avoid volatility on securities with upcoming dividends

Classic Algorithm Example

The following example algorithm shorts each equity in equal size with an upcoming dividend by the next day. It selects stocks with dividend recording over $0.5 per share to capitalize on the price shock momentum due to dividend payment.

```C#
public class UpcomingDividendsExampleAlgorithm : QCAlgorithm
{
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Universe consists of equities with upcoming dividend events.
        _universe = AddUniverse<EODHDUpcomingDividends>((dividends) => {
            // Select the stocks with upcoming dividend record date, with a sufficient dividend size.
            return from EODHDUpcomingDividends d in dividends
                    where d.DividendDate < Time.AddDays(1) && d.Dividend > 0.5m
                    select d.Symbol;
        });
       // Add a Scheduled Event to rebalance the portfolio every morning
        // based on upcoming dividend signals.
        var spy = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
        Schedule.On(DateRules.EveryDay(spy), TimeRules.AfterMarketOpen(spy, 1), Rebalance);
    }

    public void Rebalance()
    {
        // Equally invest in each member of the universe to evenly dissipate the capital risk.
        var symbols = _universe.Selected.Where(s => Securities[s].Price != 0);
        var totalCount = symbols.Count();
        var targets = symbols
            .Select(symbol => new PortfolioTarget(symbol, -1m / totalCount))
            .ToList();
        SetHoldings(targets, liquidateExistingHoldings: true);
    }
}
```

Framework Algorithm Example

The following example implements a strategy of shorting each equity in equal size with an upcoming dividend by the next day using the framework. It selects stocks with dividend recording over $0.5 per share to capitalize on the price shock momentum due to dividend payment.

```C#
public class UpcomingDividendsExampleAlgorithm : QCAlgorithm
{
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Trade on a daily basis based on upcoming dividend signals daily.
        UniverseSettings.Resolution = Resolution.Daily;
        // Universe consists of equities with upcoming dividend events.
        _universe = AddUniverse<EODHDUpcomingDividends>((dividends) => {
            // Select the stocks with upcoming dividend record dates with a sufficient dividend size.
            return from EODHDUpcomingDividends d in dividends
                    where d.DividendDate < Time.AddDays(1) && d.Dividend > 0.5m
                    select d.Symbol;
        });

        // A constant alpha model will emit insights for the stocks with dividend events in the upcoming day.
        // It is expecting a price shock due to pricing after the dividend, which might affect some momentum traders.
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Down, TimeSpan.FromDays(1)));

        // Equal weighting for each signal to dissipate the capital risk evenly.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel(Expiry.EndOfDay));
    }
}
```

Data Point Attributes

The EODHD Upcoming Dividends dataset provides `EODHDUpcomingDividends` objects, which have the following attributes:

EOD Historical Data

Upcoming Earnings

Introduction

The Upcoming Earnings dataset, provided by EOD Historical Data (EODHD), is a daily universe of US Equities with an earnings report publication in the upcoming 7 days. The data starts in January 1998 and is delivered on a daily frequency.

Compared to Nasdaq's Earning Reports as a benchmark, the Upcoming Earnings dataset has captured 96.79% of the scheduled earnings report, with a 97.25% exact-date precision of the captured events, while having a 99.28% precision within +/- 3 days. Note that the Upcoming Earnings dataset also contains unscheduled special earning reports.

For more information about the Upcoming Earnings dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

EOD Historical Data (EODHD) is a financial data provider based in France, and founded in April 2015. They focus on providing clean financial data, including stock prices, splits, dividends, fundamentals, macroeconomic indicators, technical indicators, and alternative data sources, through 24/7 API seamlessly. For more information about EODHD, visit https://eodhd.com/ .

Getting Started

The following snippet demonstrates how to request data from the Upcoming Earnings dataset:

```C#
AddUniverse<EODHDUpcomingEarnings>(SelectionFunction);
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 1998
Data Density	Sparse
Resolution	Daily
Timezone	New York

Universe Selection

To select a dynamic universe of US Equities based on the Upcoming Earnings dataset, call the `AddUniverse` method with a `EODHDUpcomingEarnings` cast.

```
C#
public override void Initialize()
{
    _universe = AddUniverse<EODHDUpcomingEarnings>(UniverseSelectionFilter);
}

private IEnumerable<Symbol> UniverseSelectionFilter(IEnumerable<EODHDUpcomingEarnings> earnings)
{
    return from d in earnings
           where d.ReportDate <= Time.AddDays(3) && d.Estimate > 0m
           select d.Symbol;
}
```

For more information about universe settings, see Settings .

Requesting Data

This dataset is designed for universe selection. However, you can add Upcoming Earnings data to your algorithm using the `AddData<EODHDUpcomingEarnings>` method.

```
C#
public class UpcomingEarningsDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);
        _symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _datasetSymbol = AddData<EODHDUpcomingEarnings>("earnings").Symbol;
    }
}
```

Accessing Data

To get the current Upcoming Earnings data, call the `Get<EODHDUpcomingEarnings>` method from the current `Slice` and index the result with the security `Symbol` . `Slice` objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    var upcomingEarnings = slice.Get<EODHDUpcomingEarnings>();
    if (upcomingEarnings.TryGetValue(_symbol, out var upcomingEarningsDataPoint))
    {
        Log($"{_symbol} will report earnings at {upcomingEarningsDataPoint.ReportDate}
{upcomingEarningsDataPoint.ReportTime} with estimated EPS {upcomingEarningsDataPoint.Estimate}");
    }
}
```

You can also iterate through all of the dataset objects in the current `Slice`

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<EODHDUpcomingEarnings>())
    {
        var equitySymbol = kvp.Key;
        var upcomingEarningsDataPoint = kvp.Value;
        Log($"{equitySymbol} will report earnings at {upcomingEarningsDataPoint.ReportDate}
{upcomingEarningsDataPoint.ReportTime} with estimated EPS {upcomingEarningsDataPoint.Estimate}");
    }
}
```

Historical Data

To get historical Upcoming Earnings for the universe, call the `History` method with the type `EODHDUpcomingEarning` .

```C#
var history = History<EODHDUpcomingEarnings>(TimeSpan.FromDays(100), Resolution.Daily);
```

To get historical Upcoming Earnings data for a known security, call the `History` method with the type `EODHDUpcomingEarning` cast and the security `Symbol` .

```C#
var history = History<EODHDUpcomingEarnings>(TimeSpan.FromDays(100), Resolution.Daily)
    .Select(x => x[_symbol])
    .ToList();
```

If there is no data in the period you request, the history result is empty. For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_datasetSymbol);
```

Example Applications

The Upcoming Earnings dataset provides timely notifications about earnings announcements, allowing traders to make capitalize on potential price movements and manage risks effectively. Examples include the following strategies:

- Short Straddle to trade on heightened volatility during earnings report.
- Filter universe for the stocks with or without upcoming earnings report to trade or avoid volatility.
- Hold the stocks with upcoming earnings estimated to be positive.

Classic Algorithm Example

The following example algorithm long a straddle on the stocks with upcoming earnings report within 5 days to trade the volatility. It allows 5 days to build up the volatility and capitalize the positions 1 day after the earnings announcement.

```C#
public class UpcomingEarningsExampleAlgorithm : QCAlgorithm
{
    // A dictionary to save the underlying-call,put pair for position open/close management.
    private Dictionary<Symbol, (Symbol Call, Symbol Put)> _optionsBySymbol = [];

    public override void Initialize()
    {
```

```
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Seed the last price as price since we need to use the underlying price for option contract filtering when
it join the universe.
        Settings.SeedInitialPrices = true;

        // Trade on daily basis based on daily upcoming earnings signals.
        UniverseSettings.Resolution = Resolution.Daily;
        // Option trading requires raw price for strike price comparison.
        UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;
        // Universe consists of equities with upcoming earnings events.
        AddUniverse<EODHDUpcomingEarnings>((earnings) => {
            foreach(var i in Enumerable.Range(1,5).Reverse())
            {
                var targetDate = Time.AddDays(i);
                var symbols = earnings
                    // We do not want to lock our fund too early, so filter for stocks is in lower volatility but
will go up.
                    // Assuming 5 days before the earnings report publish is less volatile.
                    // We do not want depository due to their low liquidity.
                    .Where(datum => (datum as EODHDUpcomingEarnings).ReportDate == targetDate)
                    .ToHashSet(datum => datum.Symbol);
                if (symbols.IsNullOrEmpty()) continue;
                Plot("Universe", "Count", symbols.Count);
                return symbols;
            }
            return [];
        });
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        // Actions only based on the equity universe changes.
        foreach (var removed in changes.RemovedSecurities.Where(x=> x.Type == SecurityType.Equity))
        {
            // Liquidate any assigned position.
            Liquidate(removed.Symbol);
            // Liquidate the option positions and capitalize the volatility 1-day after the earnings announcement.
            if (_optionsBySymbol.Remove(removed.Symbol, out var options))
            {
                RemoveOptionContract(options.Call);
                RemoveOptionContract(options.Put);
            }
        }

        // Actions only based on the equity universe changes.
        foreach (var added in changes.AddedSecurities.Where(x=> x.Type == SecurityType.Equity))
        {
            // Select the option contracts to construct a straddle to trade the volatility.
            var (call, put) = SelectOptionContracts(added.Symbol);
            if (call == null || put == null)
            {
                continue;
            }
                _optionsBySymbol[added.Symbol] = (call, put);
            // Request the option contract data for trading.
            call = AddOptionContract(call).Symbol;
            put = AddOptionContract(put).Symbol;
            // Long a straddle by shorting the selected ATM call and put.
            ComboMarketOrder(
                new List<Leg> {
                    Leg.Create(call, 1),
                    Leg.Create(put, 1)
                },
                1
            );
        }
    }

    private (Symbol, Symbol) SelectOptionContracts(Symbol underlying)
    {
        // Get all tradable option contracts for filtering.
        var optionContractList = OptionChain(underlying);

        // Expiry at least 30 days later to have a smaller theta to reduce time decay loss.
        // Yet also be ensure liquidity over the volatility fluctuation hence we take the closet expiry after that.
        var longExpiries = optionContractList.Where(x => x.ID.Date >= Time.AddDays(30)).ToList();
        if (longExpiries.Count < 2)
        {
```

```
            return (null, null);
        }
        var expiry = longExpiries.Min(x => x.ID.Date);
        var filteredContracts = optionContractList.Where(x => x.ID.Date == expiry).ToList();

        // Select ATM call and put to form a straddle for trading the volatility.
        var strike = filteredContracts.MinBy(x => Math.Abs(x.ID.StrikePrice - Securities[underlying].Price))
            .ID.StrikePrice;
        var atmContracts = filteredContracts.Where(x => x.ID.StrikePrice == strike).ToList();
        if (atmContracts.Count < 2)
        {
            return (null, null);
        }

        var atmCall = atmContracts.Single(x => x.ID.OptionRight == OptionRight.Call);
        var atmPut = atmContracts.Single(x => x.ID.OptionRight == OptionRight.Put);
        return (atmCall, atmPut);
    }
}
```

Framework Algorithm Example

The following example algorithm long a straddle on the stocks with upcoming earnings report within 5 days to trade the volatility implemented in framework algorithm. It allows 5 days to build up the volatility and capitalize the positions 1 day after the earnings announcement.

C#

```
public class UpcomingEarningsExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);

        // Seed the last price as price since we need to use the underlying price for option contract filtering when
it join the universe.
        Settings.SeedInitialPrices = true;

        // Trade on daily basis based on daily upcoming earnings signals.
        UniverseSettings.Resolution = Resolution.Daily;
        // Option trading requires raw price for strike price comparison.
        UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;
        // Universe consists of equities with upcoming earnings events.
        AddUniverse<EODHDUpcomingEarnings>((earnings) => {
            foreach(var i in Enumerable.Range(1,5).Reverse())
            {
                var targetDate = Time.AddDays(i);
                var symbols = earnings
                    // We do not want to lock our fund too early, so filter for stocks is in lower volatility but
will go up.
                    // Assuming 5 days before the earnings report publish is less volatile.
                    // We do not want depository due to their low liquidity.
                    .Where(datum => (datum as EODHDUpcomingEarnings).ReportDate == targetDate)
                    .ToHashSet(datum => datum.Symbol);
                if (symbols.IsNullOrEmpty()) continue;
                Plot("Universe", "Count", symbols.Count);
                return symbols;
            }
            return [];
        });

        // A custom alpha model that emit signal based on upcoming earnings events.
        AddAlpha(new UpcomingEarningsAlphaModel());

        // To trade 1 contract per signal to ensure equal size per leg of the short straddle.
        SetPortfolioConstruction(new SingleSharePortfolioConstructionModel());
    }
}

public class UpcomingEarningsAlphaModel : AlphaModel
{
    // A dictionary to save the underlying-call,put pair for position open/close management.
    private Dictionary<Symbol, (Symbol Call, Symbol Put)> _optionsBySymbol = [];
```

```
    public override IEnumerable<Insight> Update(QCAlgorithmFrameworkBridge algorithm, Slice data)
    {
        var insights = new List<Insight>();

        // Iterate the newly added equities to trade their volatility.
        while (_optionsBySymbol.Count > 0)
        {
            var key = _optionsBySymbol.Keys.First();
            // Obtain the selected option contracts to trade and remove the key-value pair to avoid repeat ordering.
            _optionsBySymbol.Remove(key, out var contracts);
            // Long straddle (long the ATM call and put contracts) for 7 days to trade the volatility.
            // Assuming the volatility will go up due to earning announcement and reach highest volatility at 2-3
days after the report.
            insights.AddRange([
                Insight.Price(contracts.Call, TimeSpan.FromDays(7), InsightDirection.Up),
                Insight.Price(contracts.Put, TimeSpan.FromDays(7), InsightDirection.Up)
            ]);
        }

        // Remove any assigned positions of the underlying.
        foreach (var (symbol, holding) in algorithm.Portfolio)
        {
            if (holding.Type == SecurityType.Equity && holding.Invested)
            {
                insights.Add(Insight.Price(symbol, TimeSpan.FromDays(7), InsightDirection.Flat));
            }
        }

        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        // Actions only based on the equity universe changes.
        foreach (var added in changes.AddedSecurities.Where(x=> x.Type == SecurityType.Equity))
        {
            // Select the option contracts to construct a straddle to trade the volatility.
            var (call, put) = SelectOptionContracts(algorithm, added.Symbol, algorithm.Time);
            if (call == null || put == null)
            {
                continue;
            }
            // Request the option contract data for trading.
            call = algorithm.AddOptionContract(call).Symbol;
            put = algorithm.AddOptionContract(put).Symbol;
            _optionsBySymbol[added.Symbol] = (call, put);
        }
    }

    private (Symbol, Symbol) SelectOptionContracts(QCAlgorithm algorithm, Symbol underlying, DateTime time)
    {
        // Get all tradable option contracts for filtering.
        var optionContractList = algorithm.OptionChain(underlying);

        // Expiry at least 30 days later to have a smaller theta to reduce time decay loss.
        // Yet also be ensure liquidity over the volatility fluctuation hence we take the closet expiry after that.
        var longExpiries = optionContractList.Where(x => x.ID.Date >= time.AddDays(30)).ToList();
        if (longExpiries.Count < 2)
        {
            return (null, null);
        }
        var expiry = longExpiries.Min(x => x.ID.Date);
        var filteredContracts = optionContractList.Where(x => x.ID.Date == expiry).ToList();

        // Select ATM call and put to form a straddle for trading the volatility.
        var strike = filteredContracts.MinBy(x => Math.Abs(x.ID.StrikePrice -
algorithm.Securities[underlying].Price))
            .ID.StrikePrice;
        var atmContracts = filteredContracts.Where(x => x.ID.StrikePrice == strike).ToList();
        if (atmContracts.Count < 2)
        {
            return (null, null);
        }

        var atmCall = atmContracts.Single(x => x.ID.OptionRight == OptionRight.Call);
        var atmPut = atmContracts.Single(x => x.ID.OptionRight == OptionRight.Put);
        return (atmCall, atmPut);
    }
}

public class SingleSharePortfolioConstructionModel : PortfolioConstructionModel
```

```
{
    public override IEnumerable<PortfolioTarget> CreateTargets(QCAlgorithm algorithm, Insight[] insights)
    {
        if (insights.Length == 0)
        {
            return [];
        }
        // Only a single contract to ensure both legs size is equal.
        var quantity = insights.Min(i => Math.Abs(algorithm.CalculateOrderQuantity(i.Symbol, (decimal)i.Direction)))
/ 100;

        return insights.Select(i => new PortfolioTarget(i.Symbol, quantity * (int)i.Direction));
    }
}
```

Data Point Attributes

The EODHD Upcoming Earnings dataset provides `EODHDUpcomingEarnings` objects, which have the following attributes:

To filter the EODHD Upcoming Earnings data with their reporting timing, use the `EODHD.ReportTime` enum, which have the following attributes:

EOD Historical Data

Upcoming IPOs

Introduction

The Upcoming IPOs dataset, provided by EODHD, offers daily alerts for US Equities that will start their IPOs or have any updates on their IPO registrations within the upcoming 7 days. The data started in February 2013 and is delivered on a daily basis.

`Notice that this dataset might have a +/-2 days accuracy due to the data provider.`

For more information about the Upcoming IPOs dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

EODHD was a France financial data provider founded in April 2015. They focus on providing clean financial data, including stock prices, splits, dividends, fundamentals, macroeconomy indicators, technical indicators, and alternative data sources, through 24/7 API seamlessly.

Getting Started

The following snippet demonstrates how to request data from the Upcoming IPOs dataset:

```C#
AddData<EODHDUpcomingIPOs>("ipos");
AddUniverse<EODHDUpcomingIPOs>(SelectionFunction);
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	February 2013
Data Density	Sparse
Resolution	Daily
Timezone	New York

Universe Selection

To select a dynamic universe of US Equities based on the Upcoming IPOs dataset, call the `AddUniverse` method with a `EODHDUpcomingIPOs` cast.

```C#
private static readonly List<DealType> _dealTypesWanted = new { DealType.Expected, DealType.Priced };

public override void Initialize()
{
    UniverseSettings.Asynchronous = true;
    _universe = AddUniverse<EODHDUpcomingIPOs>(UniverseSelectionFilter);
}

private IEnumerable<Symol> UniverseSelectionFilter(IEnumerable<EODHDUpcomingIPOs> ipos)
{
    // confirmed non-penny stock IPO that launches within 7 days.
    return from d in ipos
           where _dealTypesWanted.Contains(d.DealType) &&
               d.IpoDate.HasValue &&
               new[] { d.LowestPrice, d.HighestPrice, d.OfferPrice }.Where(x => x.HasValue).Min().Value > 1
           select d.Symbol;
}
```

For more information about universe settings, see Settings .

Requesting Data

To add Upcoming IPOs data to your algorithm, call the `AddData<EODHDUpcomingIPOs>` method.

```C#
public class UpcomingIPOsDataAlgorithm : QCAlgorithm
{
    private Symbol _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);

        _datasetSymbol = AddData<EODHDUpcomingIPOs>("ipos").Symbol;
    }
}
```

Accessing Data

To get the current Upcoming IPOs data, call the `Get<EODHDUpcomingIPOs>` method from the current `Slice` . Then, iterate through all of the dataset objects in the current `Slice`

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<EODHDUpcomingIPOs>())
    {
        var equitySymbol = kvp.Key;
        var upcomingIPOsDataPoint = kvp.Value;
        Log($"{equitySymbol} will start IPO at {upcomingIPOsDataPoint.IPODate} with price
{upcomingIPOsDataPoint.OfferPrice} and {upcomingIPOsDataPoint.Shares} shares");
    }
}
```

Historical Data

To get historical Upcoming IPOs data, call the `History` method with the type `EODHDUpcomingIPOs` cast and the period of request. If there is no data in the period you request, the history result is empty.

```C#
var history = History<EODHDUpcomingSplits>(TimeSpan.FromDays(100), Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```
C#
RemoveSecurity(_datasetSymbol);
```

Example Applications

The Upcoming IPOs dataset provides timely notifications about upcoming IPOs start, allowing traders to capitalize on the high volatility of new stocks. Examples include the following strategies:

- Long straddle to trade the volatility of the new IPO stock.
- Arbitration on fair price versus IPO price.
- Use SMA of IPO number to estimate the IPO trend and market popularity.

Classic Algorithm Example

Example 1: New Stock

The following example algorithm holds new stocks for its IPO day to capitalize its hype.

```
C#
public class UpcomingIPOsExampleAlgorithm : QCAlgorithm
{
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Filter for new stocks to trade their hype using EODHDUpcomingIPOs.
        _universe = AddUniverse<EODHDUpcomingIPOs>(ipos =>
            ipos.Select(x => x as EODHDUpcomingIPOs)
                .Where(x => x.IpoDate <= Time.AddDays(1) && x.Exchange == Exchange.NASDAQ)
                .Select(x => x.Symbol)
        );
    }

    public override void OnData(Slice slice)
    {
        // Invest in new stocks and trade on their first day for their hype.
        // Equally invest in each new stocks to evenly dissipate the capital risk.
        foreach (var symbol in _universe.Selected)
        {
            var security = Securities[symbol];
            if (!security.Holdings.Invested && security.Price != 0)
            {
                SetHoldings(symbol, 0.1m/_universe.Selected.Count);
            }
        }
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        // Liquidate the new stocks traded today/yesterday to capitalize the first-day hype.
        foreach (var removed in changes.RemovedSecurities)
        {
            Liquidate(removed.Symbol);
        }
    }
}
```

Example 2: Market Popularity

The following example algorithm buy QQQ if the number of IPO in Nasdaq is in increasing trend, sell otherwise.

```C#
public class IndexOptionHandlingDataAlgorithm : QCAlgorithm
{
    // Create a EMA indicator to estimate the trend of IPO number to reflect the market popularity.
    private ExponentialMovingAverage _ema = new(50);
    private Symbol _qqq;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        UniverseSettings.Resolution = Resolution.Daily;
        // Filter for new stocks to trade their hype using EODHDUpcomingIPOs.
        AddUniverse<EODHDUpcomingIPOs>(Selection);
        // Request QQQ data to trade.
        _qqq = AddEquity("QQQ", Resolution.Daily).Symbol;

        SetWarmUp(50, Resolution.Daily);
    }

    private IEnumerable<Symbol> Selection(IEnumerable<BaseData> ipos)
    {
        // Filter for the stocks that IPO starts today and traded in Nasdaq.
        var universeCount = ipos.Select(x => x as EODHDUpcomingIPOs)
            .Where(x => x.IpoDate.HasValue && x.IpoDate.Value <= Time.AddDays(1) && x.Exchange == Exchange.NASDAQ)
            .Count();
        _ema.Update(Time, Convert.ToDecimal(universeCount));
        return Universe.Unchanged;
    }

    public override void OnData(Slice slice)
    {
        if (!IsWarmingUp && _ema.IsReady)
        {
            // If the EMA is decreasing, we estimate the market popularity is decreasing, so we sell QQQ.
            if (_ema.Previous > _ema)
            {
                SetHoldings(_qqq, -1m);
            }
            // Otherwise, we estimate the market popularity is increasing, so we buy QQQ.
            else
            {
                SetHoldings(_qqq, 1m);
            }
        }
    }
}
```

Framework Algorithm Example

The following example algorithm holds new stocks for its IPO day to capitalize its hype using framework algorithm.

```C#
public class UpcomingIPOsExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Filter for new stocks to trade their hype using EODHDUpcomingIPOs.
        AddUniverse<EODHDUpcomingIPOs>(ipos =>
            ipos.Select(x => x as EODHDUpcomingIPOs)
                .Where(x => x.IpoDate <= Time.AddDays(1) && x.Exchange == Exchange.NASDAQ)
                .Select(x => x.Symbol)
        );

        // Invest in new stocks and trade on their first day for their hype.
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(1), null, null,
0.01d));

        // Equally invest in each new stocks to evenly dissipate the capital risk.
        SetPortfolioConstruction(new InsightWeightingPortfolioConstructionModel(Expiry.EndOfDay));
    }
}
```

Data Point Attributes

The EODHD Upcoming IPOs dataset provides `EODHDUpcomingIPOs` objects, which have the following attributes:

EOD Historical Data

Upcoming Splits

Introduction

The Upcoming Splits dataset, provided by EODHD, offers daily alerts for US Equities that will have a split event within the upcoming 7 days. The data starts in January 2010, and is delivered on a daily frequency.

Compared to US Equity Security Master as a benchmark, the Upcoming Splits dataset has an 85.18% coverage of all splits since 2015, while having a 92.74% precision on the exact split dates of the covered ones and a 97.18% precision within +/- 3 days.

For more information about the Upcoming Splits dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

EOD Historical Data (EODHD) is a financial data provider based in France, and founded in April 2015. They focus on providing clean financial data, including stock prices, splits, dividends, fundamentals, macroeconomic indicators, technical indicators, and alternative data sources, through 24/7 API seamlessly. For more information about EODHD, visit https://eodhd.com/ .

Getting Started

The following snippet demonstrates how to request data from the Upcoming Splits dataset:

```C#
AddUniverse<EODHDUpcomingSplits>(SelectionFunction);
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2010
Data Density	Sparse
Resolution	Daily
Timezone	New York

Universe Selection

To select a dynamic universe of US Equities based on the Upcoming Splits dataset, call the `AddUniverse` method with a `EODHDUpcomingSplits` cast.

```
C#
public override void Initialize()
{
    _universe = AddUniverse<EODHDUpcomingSplits>(UniverseSelectionFilter);
}

private IEnumerable<Symol> UniverseSelectionFilter(IEnumerable<EODHDUpcomingSplits> splits)
{
    return from d in splits
           where d.SplitDate <= Time.AddDays(3) && d.SplitFactor > 1m
           select d.Symbol;
}
```

For more information about universe settings, see Settings .

Requesting Data

To add Upcoming Splits data to your algorithm, call the `AddData<EODHDUpcomingSplits>` method.

```
C#
public class UpcomingSplitsDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);
        _symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _datasetSymbol = AddData<EODHDUpcomingSplits>("splits").Symbol;
    }
}
```

Accessing Data

To get the current Upcoming Splits data, call the `Get<EODHDUpcomingSplits>` method from the current `Slice` and index the result with the security `Symbol` . `Slice` objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your security at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    var upcomingSplits = slice.Get<EODHDUpcomingSplits>();
    if (upcomingSplits.TryGetValue(_symbol, out var upcomingSplitsDataPoint))
    {
        Log($"{_symbol} will split at {upcomingSplitsDataPoint.SplitDate} with split factor
{upcomingSplitsDataPoint.SplitFactor}");
    }
}
```

You can also iterate through all of the dataset objects in the current `Slice`

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<EODHDUpcomingSplits>())
    {
        var equitySymbol = kvp.Key;
        var upcomingSplitsDataPoint = kvp.Value;
        Log($"{equitySymbol} will splits at {upcomingSplitsDataPoint.SplitDdate} with split factor
{upcomingSplitsDataPoint.SplitFactor}");
    }
}
```

Historical Data

To get historical Upcoming Splits data, call the `History` method with the type `EODHDUpcomingSplits` cast and the period of request. If there is no data in the period you request, the history result is empty.

```C#
var history = History<EODHDUpcomingSplits>(TimeSpan.FromDays(100), Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_datasetSymbol);
```

Example Applications

The Upcoming Splits dataset provides timely notifications about upcoming share split or reverse split events, allowing traders to capitalize on potential price movements and manage risks effectively. Examples include the following strategies:

- Splits into shares with lower price will provide higher liquidity for market making algorithms.
- Select a universe of stocks with upcoming splits event and trade their volatility using options.
- Buy stocks with split factor higher than 1, sell vice versa for self-defeating ones.

Classic Algorithm Example

The following example algorithm holds each equity in equal size with upcoming splits within 7 days. It selects stocks that split up to more shares with lower price and tries to capitalize the higher demand of the more afforable stocks.

C#

```
public class UpcomingSplitsExampleAlgorithm : QCAlgorithm
{
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Universe consists of equities with upcoming earnings events.
        _universe = AddUniverse<EODHDUpcomingSplits>((splits) => {
            return splits
                // Split (more shares with lower price) will make the stock more affordable and drive up the demand.
                // Hence, include all stocks that will have a split within the next 7 days.
                // Note that spliting up the stock means the split factor > 1.
                .Where(datum => (datum as EODHDUpcomingSplits).SplitFactor > 1)
                .Select(datum => datum.Symbol);
        });
        // Add a Scheduled Event to rebalance the portfolio every morning
        // based on upcoming splits signals.
        var spy = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
        Schedule.On(DateRules.EveryDay(spy), TimeRules.AfterMarketOpen(spy, 1), Rebalance);
    }

    public void Rebalance()
    {
        // Equally invest in each member of the universe to evenly dissipate the capital risk.
        var symbols = _universe.Selected.Where(s => Securities[s].Price != 0);
        var totalCount = symbols.Count();
        var targets = symbols
            .Select(symbol => new PortfolioTarget(symbol, 1m / totalCount))
            .ToList();
        SetHoldings(targets, liquidateExistingHoldings: true);
    }
}
```

Framework Algorithm Example

The following example algorithm holds each equity in equal size with upcoming splits within 7 days using algorithm framework. It selects stocks that split up to more shares with lower price and tries to capitalize the higher demand of the more afforable stocks.

```C#
public class UpcomingSplitsExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        Settings.FreePortfolioValuePercentage = 0.1m;

        // Trade on daily basis based on daily upcoming splits signals.
        UniverseSettings.Resolution = Resolution.Daily;
        // Universe consists of equities with upcoming splits events.
        AddUniverse<EODHDUpcomingSplits>((splits) => {
            return splits.OfType<EODHDUpcomingSplits>()
                // Split (more shares with lower price) will make the stock more affordable and drive up the demand.
                // Hence, include all stocks that will have a split within the next 7 days.
                // Note that spliting up the stock means the split factor > 1.
                .Where(datum => datum.SplitFactor > 1)
                .OrderBy(datum => datum.SplitFactor)
                .Take(5).Select(datum => datum.Symbol);
        });

        // Constant alpha model to emit insights for the stocks with split up event within a week, estimating their
demand and price will go up.
        // Signal stays for 2 days to fully digest the driven up demands.
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(2)));

        // Equal weighting for each signal to evenly dissipate the capital risk.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel(Expiry.EndOfDay));
    }
}
```

Data Point Attributes

The EODHD Upcoming Splits dataset provides `EODHDUpcomingSplits` objects, which have the following attributes:

Datasets

Energy Information Administration

The Treasury Department is the executive agency responsible for promoting economic prosperity and ensuring the financial security of the United States. The Department is responsible for a wide range of activities such as advising the President on economic and financial issues, encouraging sustainable economic growth, and fostering improved governance in financial institutions. The Department of the Treasury operates and maintains systems that are critical to the nation's financial infrastructure, such as the production of coin and currency, the disbursement of payments to the American public, revenue collection, and the borrowing of funds necessary to run the federal government.

US Energy Information Administration (EIA)

Energy Information Administration

US Energy Information Administration (EIA)

Introduction

The US Energy Information Administration (EIA) datasets by the Department of the Treasury tracks national and international oil production and consumption. The data covers 190 datasets, starts in January 1991, and is delivered on a daily frequency. This dataset is created by QuantConnect processing and caching the EIA archives.

For more information about the US Energy Information Administration (EIA) dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

The Treasury Department is the executive agency responsible for promoting economic prosperity and ensuring the financial security of the United States. The Department is responsible for a wide range of activities such as advising the President on economic and financial issues, encouraging sustainable economic growth, and fostering improved governance in financial institutions. The Department of the Treasury operates and maintains systems that are critical to the nation's financial infrastructure, such as the production of coin and currency, the disbursement of payments to the American public, revenue collection, and the borrowing of funds necessary to run the federal government.

Getting Started

The following snippet demonstrates how to request data from the EIA dataset:

```C#
_datasetSymbol = AddData<USEnergy>(USEnergy.Petroleum.UnitedStates.WeeklyNetImportsOfTotalPetroleumProducts).Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 1991
Asset Coverage	190 Datasets
Data Density	Sparse
Resolution	Daily
Timezone	New York

Requesting Data

To add EIA data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```
C#
public class USEnergyDataAlgorithm: QCAlgorithm
{
    private Symbol _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);
        _datasetSymbol = AddData<USEnergy>
(USEnergy.Petroleum.UnitedStates.WeeklyNetImportsOfTotalPetroleumProducts).Symbol;
    }
}
```

Accessing Data

To get the current EIA data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var dataPoint = slice[_datasetSymbol];
        Log($"{_datasetSymbol} value at {slice.Time}: {dataPoint.Value}");
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<USEnergy>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} value at {slice.Time}: {dataPoint.Value}");
    }
}
```

Historical Data

To get historical EIA data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
var history = History<USEnergy>(_datasetSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove your subscription to EIA data, call the `RemoveSecurity` method.

```
C#
RemoveSecurity(_datasetSymbol);
```

Supported Datasets

The following table shows the accessor code you need to add each EIA dataset to your algorithm:

Symbol	Accessor Code	Description
UnitedStates		
PET.WGFRPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderAdjustedNetProductionOfFinishedMotorGasoline	U.S. Refiner and Blender Adjusted Net Production of Finished Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.WGFSTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfFinishedMotorGasoline	U.S. Ending Stocks of Finished Motor Gasoline in Thousand Barrels (Mbbl)
PET.WGFUPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyProductSuppliedOfFinishedMotorGasoline	U.S. Product Supplied of Finished Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.WCSSTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfCrudeOilInSpr	U.S. Ending Stocks of Crude Oil in SPR in Thousand Barrels (Mbbl)
PET.WDGRPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfDistillateFuelOilGreaterThan500PpmSulfur	U.S. Refiner and Blender Net Production of Distillate Fuel Oil Greater than 500 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.WDGSTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfDistillateFuelOilGreaterThan500PpmSulfur	U.S. Ending Stocks of Distillate Fuel Oil, Greater Than 500 ppm Sulfur in Thousand Barrels (Mbbl)
PET.WDIEXUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyExportsOfTotalDistillate	U.S. Exports of Total Distillate in Thousand Barrels per Day (Mbbl/d)
PET.WDIIMUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfDistillateFuelOil	U.S. Imports of Distillate Fuel Oil in Thousand Barrels per Day (Mbbl/d)
PET.WDIRPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfDistillateFuelOil	U.S. Refiner and Blender Net Production of Distillate Fuel Oil in Thousand Barrels per Day (Mbbl/d)
PET.WKJSTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfKeroseneTypeJetFuel	U.S. Ending Stocks of Kerosene-Type Jet Fuel in Thousand Barrels (Mbbl)
PET.WKJUPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyProductSuppliedOfKeroseneTypeJetFuel	U.S. Product Supplied of Kerosene-Type Jet Fuel in Thousand Barrels per Day (Mbbl/d)
PET.WGTIMUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfTotalGasoline	U.S. Imports of Total Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.WGTSTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfTotalGasoline	U.S. Ending Stocks of Total Gasoline in Thousand Barrels (Mbbl)

PET.WGIRIUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyGrossInputsIntoRefineries	U.S. Gross Inputs into Refineries in Thousand Barrels per Day (Mbbl/d)
PET.WGRIMUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfReformulatedMotorGasoline	U.S. Imports of Reformulated Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.WGRRPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfReformulatedMotorGasoline	U.S. Refiner and Blender Net Production of Reformulated Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.WGRSTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfReformulatedMotorGasoline	U.S. Ending Stocks of Reformulated Motor Gasoline in Thousand Barrels (Mbbl)
PET.WDISTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfDistillateFuelOil	U.S. Ending Stocks of Distillate Fuel Oil in Thousand Barrels (Mbbl)
PET.WDIUPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyProductSuppliedOfDistillateFuelOil	U.S. Product Supplied of Distillate Fuel Oil in Thousand Barrels per Day (Mbbl/d)
PET.WKMRPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfMilitaryKeroseneTypeJetFuel	U.S. Refiner and Blender Net Production of Military Kerosene-Type Jet Fuel in Thousand Barrels per Day (Mbbl/d)
PET.WOCLEUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyOperableCrudeOilDistillationCapacity	U. S. Operable Crude Oil Distillation Capacity in Thousand Barrels per Calendar Day (Mbbl/d)
PET.WPLSTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyPropyleneNonfuelUseStocksAtBulkTerminals	U.S. Propylene Nonfuel Use Stocks at Bulk Terminals in Thousand Barrels (Mbbl)
PET.WPRSTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfPropaneAndPropylene	U.S. Ending Stocks of Propane and Propylene in Thousand Barrels (Mbbl)
PET.WPULEUS3.W	USEnergy.Petroleum.UnitedStates.WeeklyPercentUtilizationOfRefineryOperableCapacity	U.S. Percent Utilization of Refinery Operable Capacity in Percent (%)
PET.WREEXUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyExportsOfResidualFuelOil	U.S. Exports of Residual Fuel Oil in Thousand Barrels per Day (Mbbl/d)
PET.WREIMUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfResidualFuelOil	U.S. Imports of Residual Fuel Oil in Thousand Barrels per Day (Mbbl/d)
PET.WKCRPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfCommercialKeroseneTypeJetFuel	U.S. Refiner and Blender Net Production of Commercial Kerosene-Type Jet Fuel in Thousand Barrels per Day (Mbbl/d)
PET.WKJEXUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyExportsOfKeroseneTypeJetFuel	U.S. Exports of Kerosene-Type Jet Fuel in Thousand Barrels per Day (Mbbl/d)

PET.WKJIMUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfKeroseneTypeJetFuel	U.S. Imports of Kerosene-Type Jet Fuel in Thousand Barrels per Day (Mbbl/d)
PET.WKJRPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfKeroseneTypeJetFuel	U.S. Refiner and Blender Net Production of Kerosene-Type Jet Fuel in Thousand Barrels per Day (Mbbl/d)
PET.WCESTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksExcludingSprOfCrudeOil	U.S. Ending Stocks excluding SPR of Crude Oil in Thousand Barrels (Mbbl)
PET.WCREXUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyExportsOfCrudeOil	U.S. Exports of Crude Oil in Thousand Barrels per Day (Mbbl/d)
PET.WCRFPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyFieldProductionOfCrudeOil	U.S. Field Production of Crude Oil in Thousand Barrels per Day (Mbbl/d)
PET.WCRIMUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfCrudeOil	U.S. Imports of Crude Oil in Thousand Barrels per Day (Mbbl/d)
PET.WCRNTUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyNetImportsOfCrudeOil	U.S. Net Imports of Crude Oil in Thousand Barrels per Day (Mbbl/d)
PET.WCRRIUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetInputOfCrudeOil	U.S. Refiner Net Input of Crude Oil in Thousand Barrels per Day (Mbbl/d)
PET.WRERPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfResidualFuelOil	U.S. Refiner and Blender Net Production of Residual Fuel Oil in Thousand Barrels per Day (Mbbl/d)
PET.WRESTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfResidualFuelOil	U.S. Ending Stocks of Residual Fuel Oil in Thousand Barrels (Mbbl)
PET.WREUPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyProductSuppliedOfResidualFuelOil	U.S. Product Supplied of Residual Fuel Oil in Thousand Barrels per Day (Mbbl/d)
PET.WRPEXUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyExportsOfTotalPetroleumProducts	U.S. Exports of Total Petroleum Products in Thousand Barrels per Day (Mbbl/d)
PET.WRPIMUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfTotalPetroleumProducts	U.S. Imports of Total Petroleum Products in Thousand Barrels per Day (Mbbl/d)
PET.WRPNTUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyNetImportsOfTotalPetroleumProducts	U.S. Net Imports of Total Petroleum Products in Thousand Barrels per Day (Mbbl/d)
PET.WRPUPUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyProductSuppliedOfPetroleumProducts	U.S. Product Supplied of Petroleum Products in Thousand Barrels per Day (Mbbl/d)
PET.WTESTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksExcludingSprOfCrudeOilAndPetroleumProducts	U.S. Ending Stocks excluding SPR of Crude Oil and Petroleum Products in Thousand Barrels (Mbbl)

PET.WTTEXUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyExportsOfCrudeOilAndPetroleumProducts	U.S. Exports of Crude Oil and Petroleum Products in Thousand Barrels per Day (Mbbl/d)
PET.WTTIMUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfCrudeOilAndPetroleumProducts	U.S. Imports of Crude Oil and Petroleum Products in Thousand Barrels per Day (Mbbl/d)
PET.WTTNTUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyNetImportsOfCrudeOilAndPetroleumProducts	U.S. Net Imports of Crude Oil and Petroleum Products in Thousand Barrels per Day (Mbbl/d)
PET.WTTSTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfCrudeOilAndPetroleumProducts	U.S. Ending Stocks of Crude Oil and Petroleum Products in Thousand Barrels (Mbbl)
PET.WUOSTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfUnfinishedOils	U.S. Ending Stocks of Unfinished Oils in Thousand Barrels (Mbbl)
PET.WG6TP_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfOtherFinishedConventionalMotorGasoline	U.S. Refiner and Blender Net Production of Other Finished Conventional Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.WD0TP_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfDistillateFuelOil0To15PpmSulfur	U.S. Refiner and Blender Net Production of Distillate Fuel Oil, 0 to 15 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.WD1ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfDistillateFuelOilGreaterThan15To500PpmSulfur	U.S. Ending Stocks of Distillate Fuel Oil, Greater than 15 to 500 ppm Sulfur in Thousand Barrels (Mbbl)
PET.WD1TP_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyProductionOfDistillateFuelOilGreaterThan15To500PpmSulfur	U.S. Production of Distillate Fuel Oil, Greater than 15 to 500 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.WG1ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfReformulatedMotorGasolineWithFuelAlcohol	U.S. Ending Stocks of Reformulated Motor Gasoline with Fuel ALcohol in Thousand Barrels (Mbbl)
PET.WCRSTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfCrudeOil	U.S. Ending Stocks of Crude Oil in Thousand Barrels (Mbbl)
PET.WCSIMUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyCrudeOilImportsBySpr	U.S. Crude Oil Imports by SPR in Thousand Barrels per Day (Mbbl/d)
PET.WBCIMUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfGasolineBlendingComponents	U.S. Imports of Gasoline Blending Components in Thousand Barrels per Day (Mbbl/d)
PET.WBCSTUS1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfGasolineBlendingComponents	U.S. Ending Stocks of Gasoline Blending Components in Thousand Barrels (Mbbl)
PET.WCEIMUS2.W	USEnergy.Petroleum.UnitedStates.WeeklyCommercialCrudeOilImportsExcludingSpr	U.S. Commercial Crude Oil Imports Excluding SPR in Thousand Barrels per Day (Mbbl/d)

PET.WPRTP_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerBlenderAndGasPlantNetProductionOfPropaneAndPropylene	U.S. Refiner, Blender, and Gas Plant Net Production of Propane and Propylene in Thousand Barrels per Day (Mbbl/d)
PET.WG1TP_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfFinishedReformulatedMotorGasolineWithEthanol	U.S. Refiner and Blender Net Production of Finished Reformulated Motor Gasoline with Ethanol in Thousand Barrels per Day (Mbbl/d)
PET.WG3ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfReformulatedMotorGasolineNonOxygentated	U.S. Ending Stocks of Reformulated Motor Gasoline, Non-Oxygentated in Thousand Barrels (Mbbl)
PET.WG4ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfConventionalMotorGasoline	U.S. Ending Stocks of Conventional Motor Gasoline in Thousand Barrels (Mbbl)
PET.WG4TP_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfConventionalMotorGasoline	U.S. Refiner and Blender Net Production of Conventional Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.WG5ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfConventionalMotorGasolineWithFuelEthanol	U.S. Ending Stocks of Conventional Motor Gasoline with Fuel Ethanol in Thousand Barrels (Mbbl)
PET.WG5TP_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfFinishedConventionalMotorGasolineWithEthanol	U.S. Refiner and Blender Net Production of Finished Conventional Motor Gasoline with Ethanol in Thousand Barrels per Day (Mbbl/d)
PET.WG6ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfOtherConventionalMotorGasoline	U.S. Ending Stocks of Other Conventional Motor Gasoline in Thousand Barrels (Mbbl)
PET.WO6RI_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetInputOfConventionalCbobGasolineBlendingComponents	U.S. Refiner and Blender Net Input of Conventional CBOB Gasoline Blending Components in Thousand Barrels per Day (Mbbl/d)
PET.WO6ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfConventionalCbobGasolineBlendingComponents	U.S. Ending Stocks of Conventional CBOB Gasoline Blending Components in Thousand Barrels (Mbbl)
PET.WO7RI_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetInputOfConventionalGtabGasolineBlendingComponents	U.S. Refiner and Blender Net Input of Conventional GTAB Gasoline Blending Components in Thousand Barrels per Day (Mbbl/d)
PET.WO7ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfConventionalGtabGasolineBlendingComponents	U.S. Ending Stocks of Conventional GTAB Gasoline Blending Components in Thousand Barrels (Mbbl)
PET.WO9RI_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetInputOfConventionalOtherGasolineBlendingComponents	U.S. Refiner and Blender Net Input of Conventional Other Gasoline Blending Components in Thousand Barrels per Day (Mbbl/d)

PET.WO9ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfConventionalOtherGasolineBlendingComponents	U.S. Ending Stocks of Conventional Other Gasoline Blending Components in Thousand Barrels (Mbbl)
PET.W_EPD2F_PWR_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyNo2HeatingOilWholesaleResalePrice	U.S. No. 2 Heating Oil Wholesale/Resale Price in Dollars per Gallon ($/gal)
PET.W_EPC0_SKA_NUS_MBBL.W	USEnergy.Petroleum.UnitedStates.WeeklyCrudeOilStocksInTransitOnShipsFromAlaska	U.S. Crude Oil Stocks in Transit (on Ships) from Alaska in Thousand Barrels (Mbbl)
PET.W_EPC0_VSD_NUS_DAYS.W	USEnergy.Petroleum.UnitedStates.WeeklyDaysOfSupplyOfCrudeOilExcludingSpr	U.S. Days of Supply of Crude Oil excluding SPR in Number of Days (Days)
PET.W_EPD0_VSD_NUS_DAYS.W	USEnergy.Petroleum.UnitedStates.WeeklyDaysOfSupplyOfTotalDistillate	U.S. Days of Supply of Total Distillate in Number of Days (Days)
PET.W_EPD2F_PRS_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyWeeklyNo2HeatingOilResidentialPrice	U.S. Weekly No. 2 Heating Oil Residential Price in Dollars per Gallon ($/gal)
PET.WPRUP_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyProductSuppliedOfPropaneAndPropylene	U.S. Product Supplied of Propane and Propylene in Thousand Barrels per Day (Mbbl/d)
PET.WWOUP_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyProductSuppliedOfOtherOils	U.S. Product Supplied of Other Oils in Thousand Barrels per Day (Mbbl/d)
PET.WBCRI_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetInputOfGasolineBlendingComponents	U.S. Refiner and Blender Net Input of Gasoline Blending Components in Thousand Barrels per Day (Mbbl/d)
PET.WD0ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfDistillateFuelOil0To15PpmSulfur	U.S. Ending Stocks of Distillate Fuel Oil, 0 to 15 ppm Sulfur in Thousand Barrels (Mbbl)
PET.W_EPJK_VSD_NUS_DAYS.W	USEnergy.Petroleum.UnitedStates.WeeklyDaysOfSupplyOfKeroseneTypeJetFuel	U.S. Days of Supply of Kerosene-Type Jet Fuel in Number of Days (Days)
PET.W_EPM0_VSD_NUS_DAYS.W	USEnergy.Petroleum.UnitedStates.WeeklyDaysOfSupplyOfTotalGasoline	U.S. Days of Supply of Total Gasoline in Number of Days (Days)
PET.W_EPPA_SAE_NUS_MBBL.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfAsphaltAndRoadOil	U.S. Ending Stocks of Asphalt and Road Oil in Thousand Barrels (Mbbl)
PET.W_EPPK_SAE_NUS_MBBL.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfKerosene	U.S. Ending Stocks of Kerosene in Thousand Barrels (Mbbl)
PET.W_EPDM10_VUA_NUS_2.W	USEnergy.Petroleum.UnitedStates.WeeklySupplyAdjustmentOfDistillateFuelOilGreaterThan15To500PpmSulfur	U.S. Supply Adjustment of Distillate Fuel Oil, Greater than 15 to 500 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)

PET.WG5IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfConventionalMotorGasolineWithFuelEthanol	U.S. Imports of Conventional Motor Gasoline with Fuel Ethanol in Thousand Barrels per Day (Mbbl/d)
PET.WG6IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfOtherConventionalMotorGasoline	U.S. Imports of Other Conventional Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.WD0IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfDistillateFuelOil0To15PpmSulfur	U.S. Imports of Distillate Fuel Oil, 0 to 15 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.WD1IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfDistillateFuelOilGreaterThan15To500PpmSulfur	U.S. Imports of Distillate Fuel Oil, Greater than 15 to 500 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.WD2IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfDistillateFuelOilGreaterThan500To2000PpmSulfur	U.S. Imports of Distillate Fuel Oil, Greater than 500 to 2000 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.WPRIM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfPropaneAndPropylene	U.S. Imports of Propane and Propylene in Thousand Barrels per Day (Mbbl/d)
PET.WO7IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfConventionalGtabGasolineBlendingComponents	U.S. Imports of Conventional GTAB Gasoline Blending Components in Thousand Barrels per Day (Mbbl/d)
PET.WD3IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfDistillateFuelOilGreaterThan2000PpmSulfur	U.S. Imports of Distillate Fuel Oil, Greater than 2000 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.WG1IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfReformulatedMotorGasolineWithFuelAlcohol	U.S. Imports of Reformulated Motor Gasoline with Fuel ALcohol in Thousand Barrels per Day (Mbbl/d)
PET.WG4IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfConventionalMotorGasoline	U.S. Imports of Conventional Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.WO9IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfConventionalOtherGasolineBlendingComponents	U.S. Imports of Conventional Other Gasoline Blending Components in Thousand Barrels per Day (Mbbl/d)
PET.WO6IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfConventionalCbobGasolineBlendingComponents	U.S. Imports of Conventional CBOB Gasoline Blending Components in Thousand Barrels per Day (Mbbl/d)
PET.W_EPPK_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfKerosene	U.S. Blender Net Production of Kerosene in Thousand Barrels per Day (Mbbl/d)
PET.W_EPPK_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfKerosene	U.S. Refiner Net Production of Kerosene in Thousand Barrels per Day (Mbbl/d)
PET.W_EPPO6_SAE_NUS_MBBL.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfOtherOilsExcludingFuelEthanol	U.S. Ending Stocks of Other Oils (Excluding Fuel Ethanol) in Thousand Barrels (Mbbl)

PET.W_EPPR_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfResidualFuelOil	U.S. Refiner Net Production of Residual Fuel Oil in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0R_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfReformulatedMotorGasoline	U.S. Blender Net Production of Reformulated Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0R_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfReformulatedMotorGasoline	U.S. Refiner Net Production of Reformulated Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPOOXE_SAE_NUS_MBBL.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfFuelEthanol	U.S. Ending Stocks of Fuel Ethanol in Thousand Barrels (Mbbl)
PET.W_EPD0_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfDistillateFuelOil	U.S. Blender Net Production of Distillate Fuel Oil in Thousand Barrels per Day (Mbbl/d)
PET.W_EPD0_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfDistillateFuelOil	U.S. Refiner Net Production of Distillate Fuel Oil in Thousand Barrels per Day (Mbbl/d)
PET.W_EPJK_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfKeroseneTypeJetFuel	U.S. Blender Net Production of Kerosene-Type Jet Fuel in Thousand Barrels per Day (Mbbl/d)
PET.W_EPJK_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfKeroseneTypeJetFuel	U.S. Refiner Net Production of Kerosene-Type Jet Fuel in Thousand Barrels per Day (Mbbl/d)
PET.W_EPLLPA_PRS_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyPropaneResidentialPrice	U.S. Propane Residential Price in Dollars per Gallon ($/gal)
PET.W_EPLLPA_PWR_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyPropaneWholesaleResalePrice	U.S. Propane Wholesale/Resale Price in Dollars per Gallon ($/gal)
PET.W_EPOBGRR_YIR_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetInputOfMotorGasolineBlendingComponentsRbob	U.S. Refiner and Blender Net Input of Motor Gasoline Blending Components, RBOB in Thousand Barrels per Day (Mbbl/d)
PET.W_EPL0XP_SAE_NUS_MBBL.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfNgplsLrgsExcludingPropanePropylene	U.S. Ending Stocks of NGPLs/LRGs (Excluding Propane/Propylene) in Thousand Barrels (Mbbl)
PET.W_EPLLPZ_VSD_NUS_DAYS.W	USEnergy.Petroleum.UnitedStates.WeeklyDaysOfSupplyOfPropanePropylene	U.S. Days of Supply of Propane/Propylene in Number of Days (Days)
PET.W_EPM0C_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfConventionalMotorGasoline	U.S. Blender Net Production of Conventional Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0C_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfConventionalMotorGasoline	U.S. Refiner Net Production of Conventional Motor Gasoline in Thousand Barrels per Day (Mbbl/d)

PET.W_EPM0F_VUA_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklySupplyAdjustmentOfFinishedMotorGasoline	U.S. Supply Adjustment of Finished Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0F_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfFinishedMotorGasoline	U.S. Blender Net Production of Finished Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0F_YPR_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfFinishedMotorGasoline	U.S. Refiner and Blender Net Production of Finished Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0F_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfFinishedMotorGasoline	U.S. Refiner Net Production of Finished Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPD00H_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfDistillateFuelOilGreaterThan500PpmSulfur	U.S. Blender Net Production of Distillate Fuel Oil, Greater Than 500 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.W_EPD00H_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfDistillateFuelOilGreaterThan500PpmSulfur	U.S. Refiner Net Production of Distillate Fuel Oil, Greater Than 500 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.W_EPDM10_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfDistillateFuelOilGreaterThan15To500PpmSulfur	U.S. Blender Net Production of Distillate Fuel Oil, Greater than 15 to 500 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.W_EPDM10_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfDistillateFuelOilGreaterThan15To500PpmSulfur	U.S. Refiner Net Production of Distillate Fuel Oil, Greater than 15 to 500 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.W_EPDXL0_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfDistillateFuelOil0To15PpmSulfur	U.S. Blender Net Production of Distillate Fuel Oil, 0 to 15 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.W_EPDXL0_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfDistillateFuelOil0To15PpmSulfur	U.S. Refiner Net Production of Distillate Fuel Oil, 0 to 15 ppm Sulfur in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CA_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfConventionalMotorGasolineWithFuelEthanol	U.S. Blender Net Production of Conventional Motor Gasoline with Fuel Ethanol in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CA_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfConventionalMotorGasolineWithFuelEthanol	U.S. Refiner Net Production of Conventional Motor Gasoline with Fuel Ethanol in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CO_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfOtherConventionalMotorGasoline	U.S. Blender Net Production of Other Conventional Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CO_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfOtherConventionalMotorGasoline	U.S. Refiner Net Production of Other Conventional Motor Gasoline in Thousand Barrels per Day (Mbbl/d)

PET.W_EPM0RA_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfReformulatedMotorGasolineWithFuelAlcohol	U.S. Blender Net Production of Reformulated Motor Gasoline with Fuel ALcohol in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0RA_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfReformulatedMotorGasolineWithFuelAlcohol	U.S. Refiner Net Production of Reformulated Motor Gasoline with Fuel ALcohol in Thousand Barrels per Day (Mbbl/d)
PET.W_EPOOXE_YOP_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyOxygenatePlantProductionOfFuelEthanol	U.S. Oxygenate Plant Production of Fuel Ethanol in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CAL55_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfMotorGasolineFinishedConventionalEd55AndLower	U.S. Blender Net Production of Motor Gasoline, Finished, Conventional, Ed55 and Lower in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CAL55_YPT_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfFinishedConventionalMotorGasolineEd55AndLower	U.S. Refiner and Blender Net Production of Finished Conventional Motor Gasoline, Ed 55 and Lower in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CAL55_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfMotorGasolineFinishedConventionalEd55AndLower	U.S. Refiner Net Production of Motor Gasoline, Finished, Conventional, Ed55 and Lower in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0F_EEX_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyExportsOfFinishedMotorGasoline	U.S. Exports of Finished Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0F_IM0_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfFinishedMotorGasoline	U.S. Imports of Finished Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0RO_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfOtherReformulatedMotorGasoline	U.S. Blender Net Production of Other Reformulated Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0RO_YPT_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfOtherFinishedReformulatedMotorGasoline	U.S. Refiner and Blender Net Production of Other Finished Reformulated Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0RO_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfOtherReformulatedMotorGasoline	U.S. Refiner Net Production of Other Reformulated Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPOBGRR_SAE_NUS_MBBL.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfMotorGasolineBlendingComponentsRbob	U.S. Ending Stocks of Motor Gasoline Blending Components, RBOB in Thousand Barrels (Mbbl)
PET.W_EPOOXE_YIR_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetInputOfFuelEthanol	U.S. Refiner and Blender Net Input of Fuel Ethanol in Thousand Barrels per Day (Mbbl/d)

PET.W_EPM0CAG55_IM0_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfMotorGasolineFinishedConventionalGreaterThanEd55	U.S. Imports of Motor Gasoline, Finished, Conventional, Greater than Ed55 in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CAL55_IM0_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfMotorGasolineFinishedConventionalEd55AndLower	U.S. Imports of Motor Gasoline, Finished, Conventional, Ed55 and Lower in Thousand Barrels per Day (Mbbl/d)
PET.W_EPC0_IMU_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyCrudeOilImportsForSprByOthers	U.S. Crude Oil Imports for SPR by Others in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CAG55_SAE_NUS_MBBL.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfConventionalMotorGasolineGreaterThanEd55	U.S. Ending Stocks of Conventional Motor Gasoline, Greater than Ed55 in Thousand Barrels (Mbbl)
PET.W_EPOOXE_IM0_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfFuelEthanol	U.S. Imports of Fuel Ethanol in Thousand Barrels per Day (Mbbl/d)
PET.W_EPL0XP_IM0_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfLiquefiedPetroleumGassesLessPropanePropylene	U.S. Imports of Liquefied Petroleum Gasses Less Propane/Propylene in Thousand Barrels per Day (Mbbl/d)
PET.W_EPLLPZ_EEX_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyExportsOfPropaneAndPropylene	U.S. Exports of Propane and Propylene in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0RO_IM0_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfOtherReformulatedMotorGasoline	U.S. Imports of Other Reformulated Motor Gasoline in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CAG55_YPB_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyBlenderNetProductionOfMotorGasolineFinishedConventionalGreaterThanEd55	U.S. Blender Net Production of Motor Gasoline, Finished, Conventional, Greater Than Ed55 in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CAG55_YPT_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerAndBlenderNetProductionOfFinishedConventionalMotorGasolineGreaterThanEd55	U.S. Refiner and Blender Net Production of Finished Conventional Motor Gasoline, Greater than Ed 55 in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CAG55_YPY_NUS_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyRefinerNetProductionOfFinishedConventionalMotorGasolineGreaterThanEd55	U.S. Refiner Net Production of Finished Conventional Motor Gasoline, Greater than Ed 55 in Thousand Barrels per Day (Mbbl/d)
PET.W_EPM0CAL55_SAE_NUS_MBBL.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfConventionalMotorGasolineEd55AndLower	U.S. Ending Stocks of Conventional Motor Gasoline, Ed55 and Lower in Thousand Barrels (Mbbl)
PET.W_EPPK_IM0_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfKerosene	U.S. Imports of Kerosene in Thousand Barrels per Day (Mbbl/d)
PET.W_EPPO4_EEX_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyExportsOfOtherOils	U.S. Exports of Other Oils in Thousand Barrels per Day (Mbbl/d)
PET.W_EPPO6_IM0_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfOtherOilsExcludingFuelEthanol	U.S. Imports of Other Oils (Excluding Fuel Ethanol) in Thousand Barrels per Day (Mbbl/d)

PET.W_EPOBGRR_IM0_NUS-Z00_MBBLD.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsFromAllCountriesOfMotorGasolineBlendingComponentsRbob	U.S. Imports from All Countries of Motor Gasoline Blending Components, RBOB in Thousand Barrels per Day (Mbbl/d)
PET.EMM_EPMR_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyRegularAllFormulationsRetailGasolinePrices	U.S. Regular All Formulations Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMM_EPMM_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyMidgradeAllFormulationsRetailGasolinePrices	U.S. Midgrade All Formulations Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMM_EPMP_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyPremiumAllFormulationsRetailGasolinePrices	U.S. Premium All Formulations Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMM_EPM0_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyAllGradesAllFormulationsRetailGasolinePrices	U.S. All Grades All Formulations Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMM_EPM0R_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyAllGradesReformulatedRetailGasolinePrices	U.S. All Grades Reformulated Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMM_EPMMR_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyMidgradeReformulatedRetailGasolinePrices	U.S. Midgrade Reformulated Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMM_EPMPR_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyPremiumReformulatedRetailGasolinePrices	U.S. Premium Reformulated Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMM_EPMRU_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyRegularConventionalRetailGasolinePrices	U.S. Regular Conventional Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMM_EPMRR_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyRegularReformulatedRetailGasolinePrices	U.S. Regular Reformulated Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMD_EPD2D_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyNo2DieselRetailPrices	U.S. No 2 Diesel Retail Prices in Dollars per Gallon ($/gal)
PET.EMM_EPMPU_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyPremiumConventionalRetailGasolinePrices	U.S. Premium Conventional Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMM_EPMMU_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyMidgradeConventionalRetailGasolinePrices	U.S. Midgrade Conventional Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMM_EPM0U_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyAllGradesConventionalRetailGasolinePrices	U.S. All Grades Conventional Retail Gasoline Prices in Dollars per Gallon ($/gal)
PET.EMD_EPD2DXL0_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyNo2DieselUltraLowSulfur015PpmRetailPrices	U.S. No 2 Diesel Ultra Low Sulfur (0-15 ppm) Retail Prices in Dollars per Gallon ($/gal)

PET.W_EPC0_SAX_NUS_MBBL.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksExcludingSprAndIncludingLeaseStockOfCrudeOil	U.S. Ending Stocks excluding SPR and including Lease Stock of Crude Oil in Thousand Barrels (Mbbl)
PET.EMD_EPD2DM10_PTE_NUS_DPG.W	USEnergy.Petroleum.UnitedStates.WeeklyNo2DieselLowSulfur15500PpmRetailPrices	U.S. No 2 Diesel Low Sulfur (15-500 ppm) Retail Prices in Dollars per Gallon ($/gal)
PET.WO3IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfReformulatedRbobWithAlcoholGasolineBlendingComponents	U.S. Imports of Reformulated RBOB with Alcohol Gasoline Blending Components in Thousand Barrels per Day (Mbbl/d)
PET.WO4IM_NUS-Z00_2.W	USEnergy.Petroleum.UnitedStates.WeeklyImportsOfReformulatedRbobWithEtherGasolineBlendingComponents	U.S. Imports of Reformulated RBOB with Ether Gasoline Blending Components in Thousand Barrels per Day (Mbbl/d)
PET.WO2ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfReformulatedGtabGasolineBlendingComponents	U.S. Ending Stocks of Reformulated GTAB Gasoline Blending Components in Thousand Barrels (Mbbl)
PET.WO3ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfReformulatedRbobWithAlcoholGasolineBlendingComponents	U.S. Ending Stocks of Reformulated RBOB with Alcohol Gasoline Blending Components in Thousand Barrels (Mbbl)
PET.WO4ST_NUS_1.W	USEnergy.Petroleum.UnitedStates.WeeklyEndingStocksOfReformulatedRbobWithEtherGasolineBlendingComponents	U.S. Ending Stocks of Reformulated RBOB with Ether Gasoline Blending Components in Thousand Barrels (Mbbl)
EquatorialGuinea		
PET.W_EPC0_IM0_NUS-NEK_MBBLD.W	USEnergy.Petroleum.EquatorialGuinea.WeeklyImportsFromEquatorialGuineaOfCrudeOil	U.S. Imports from Equatorial Guinea of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Iraq		
PET.W_EPC0_IM0_NUS-NIZ_MBBLD.W	USEnergy.Petroleum.Iraq.WeeklyImportsFromIraqOfCrudeOil	U.S. Imports from Iraq of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Kuwait		
PET.W_EPC0_IM0_NUS-NKU_MBBLD.W	USEnergy.Petroleum.Kuwait.WeeklyImportsFromKuwaitOfCrudeOil	U.S. Imports from Kuwait of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Mexico		
PET.W_EPC0_IM0_NUS-NMX_MBBLD.W	USEnergy.Petroleum.Mexico.WeeklyImportsFromMexicoOfCrudeOil	U.S. Imports from Mexico of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Nigeria		
PET.W_EPC0_IM0_NUS-NNI_MBBLD.W	USEnergy.Petroleum.Nigeria.WeeklyImportsFromNigeriaOfCrudeOil	U.S. Imports from Nigeria of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Norway		

PET.W_EPC0_IM0_NUS-NNO_MBBLD.W	USEnergy.Petroleum.Norway.WeeklyImportsFromNorwayOfCrudeOil	U.S. Imports from Norway of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Russia		
PET.W_EPC0_IM0_NUS-NRS_MBBLD.W	USEnergy.Petroleum.Russia.WeeklyImportsFromRussiaOfCrudeOil	U.S. Imports from Russia of Crude Oil in Thousand Barrels per Day (Mbbl/d)
SaudiArabia		
PET.W_EPC0_IM0_NUS-NSA_MBBLD.W	USEnergy.Petroleum.SaudiArabia.WeeklyImportsFromSaudiArabiaOfCrudeOil	U.S. Imports from Saudi Arabia of Crude Oil in Thousand Barrels per Day (Mbbl/d)
UnitedKingdom		
PET.W_EPC0_IM0_NUS-NUK_MBBLD.W	USEnergy.Petroleum.UnitedKingdom.WeeklyImportsFromUnitedKingdomOfCrudeOil	U.S. Imports from United Kingdom of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Venezuela		
PET.W_EPC0_IM0_NUS-NVE_MBBLD.W	USEnergy.Petroleum.Venezuela.WeeklyImportsFromVenezuelaOfCrudeOil	U.S. Imports from Venezuela of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Algeria		
PET.W_EPC0_IM0_NUS-NAG_MBBLD.W	USEnergy.Petroleum.Algeria.WeeklyImportsFromAlgeriaOfCrudeOil	U.S. Imports from Algeria of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Angola		
PET.W_EPC0_IM0_NUS-NAO_MBBLD.W	USEnergy.Petroleum.Angola.WeeklyImportsFromAngolaOfCrudeOil	U.S. Imports from Angola of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Brazil		
PET.W_EPC0_IM0_NUS-NBR_MBBLD.W	USEnergy.Petroleum.Brazil.WeeklyImportsFromBrazilOfCrudeOil	U.S. Imports from Brazil of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Canada		
PET.W_EPC0_IM0_NUS-NCA_MBBLD.W	USEnergy.Petroleum.Canada.WeeklyImportsFromCanadaOfCrudeOil	U.S. Imports from Canada of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Congo		
PET.W_EPC0_IM0_NUS-NCF_MBBLD.W	USEnergy.Petroleum.Congo.WeeklyImportsFromCongoBrazzavilleOfCrudeOil	U.S. Imports from Congo (Brazzaville) of Crude Oil in Thousand Barrels per Day (Mbbl/d)
Colombia		
PET.W_EPC0_IM0_NUS-NCO_MBBLD.W	USEnergy.Petroleum.Colombia.WeeklyImportsFromColombiaOfCrudeOil	U.S. Imports from Colombia of Crude Oil in Thousand Barrels per Day (Mbbl/d)

Ecuador		
PET.W_EPC0_IM0_NUS-NEC_MBBLD.W	USEnergy.Petroleum.Ecuador.WeeklyImportsFromEcuadorOfCrudeOil	U.S. Imports from Ecuador of Crude Oil in Thousand Barrels per Day (Mbbl/d)

Example Applications

The EIA dataset enables you to monitor national and international oil production and consumption in you trading strategies. Examples include the following strategies:

- Trading petroleum companies when there is a change in net imports of petroleum products
- Trading country ETFs when there is a change in the country's net import of resources
- Adjusting exposure to vehicle manufacturer stocks when the supply of gasoline is higher/lower than historical levels

Classic Algorithm Example

The following example algorithm buy a petroleum company when net imports of petroleum products are increasing. When the net imports are decreasing, it short sells the company.

C#

```
public class USEnergyDataAlgorithm : QCAlgorithm
{
    private decimal? previousValue;
    private Symbol tradableSymbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Requesting data, we trade Exxon Mobil Corp (XOM) as the proxy of petroleum investment
        tradableSymbol = AddEquity("XOM", Resolution.Daily).Symbol;
        // Request US weekly net import petroleum products data for trade signal generation
        var USEnergySymbol = AddData<USEnergy>
(USEnergy.Petroleum.UnitedStates.WeeklyNetImportsOfTotalPetroleumProducts).Symbol;

        // Historical data
        var history = History<USEnergy>(USEnergySymbol, 60, Resolution.Daily);
        Debug($"We got {history.Count()} items from our history request");

        // Get latest value for net imports of petroleum products for trade system readiness
        previousValue = history.Last().Value;
    }


    public override void OnData(Slice slice)
    {
        // Trade based on the current net imports of petroleum products
        var points = slice.Get<USEnergy>();
        decimal? currentValue = null;
        foreach (var point in points.Values)
        {
            currentValue = point.Value;
        }
        if (currentValue == null)
        {
            return;
        }
        Plot("Net Imports", "Petroleum", (double)currentValue);

        // Buy when net imports of petroleum products are increasing, assuming the demand increases that bring up the
petroleum price
        // Hence the inventory and revenue of the proxy increase, supporting the stock price goes up
        if (currentValue > previousValue)
        {
            SetHoldings(tradableSymbol, 1);
        }

        //  Short sell when net imports of petroleum products are decreasing, decreasing demand lowers the petroleum
price and the proxy stock price
        if (currentValue < previousValue)
        {
            SetHoldings(tradableSymbol, -1);
        }

        previousValue = currentValue;
    }
}
```

Framework Algorithm Example

The following example algorithm buy a petroleum company when net imports of petroleum products are increasing. When the net imports are decreasing, it short sells the company.

C#

```
public class USEnergyDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        UniverseSettings.Resolution = Resolution.Daily;
```

```
        UniverseSettings.Resolution = Resolution.Daily;
        // We only trade Exxon Mobil Corp (XOM) as the proxy of petroleum investment
        var symbols = new[] {QuantConnect.Symbol.Create("XOM", SecurityType.Equity, Market.USA)};
        AddUniverseSelection(new ManualUniverseSelectionModel(symbols));
        // A custom alpha model that emit trade insights based on US Energy data
        AddAlpha(new USEnergyAlphaModel(this));
        // Equally invest to dissipate non-systematic capital concentration risk on individual stock
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }
}


public class USEnergyAlphaModel : AlphaModel
{
    private Symbol USEnergySymbol;
    private List<Symbol> tradableSymbols = new List<Symbol>();
    private decimal? previousValue;

    public USEnergyAlphaModel(QCAlgorithm algorithm)
    {
        // Request US weekly net import petroleum products data for trade signal generation
        USEnergySymbol = algorithm.AddData<USEnergy>
(USEnergy.Petroleum.UnitedStates.WeeklyNetImportsOfTotalPetroleumProducts).Symbol;

        // Historical data
        var history = algorithm.History<USEnergy>(USEnergySymbol, 60, Resolution.Daily);
        algorithm.Debug($"We got {history.Count()} items from our history request");
        // Get latest value for net imports of petroleum products for trade system readiness
        previousValue = history.Last().Value;
    }

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = new List<Insight>();

        // Trade based on the current net imports of petroleum products
        var points = slice.Get<USEnergy>();
        decimal? currentValue = null;
        foreach (var point in points.Values)
        {
            currentValue = point.Value;
        }
        if (currentValue == null)
        {
            return insights;
        }

        foreach (var symbol in tradableSymbols)
        {
            // Buy when net imports of petroleum products are increasing, assuming the demand increases that bring up
the petroleum price
            // Hence the inventory and revenue of the proxy increase, supporting the stock price goes up
            if (currentValue > previousValue)
            {
                insights.Add(Insight.Price(symbol, TimeSpan.FromDays(7),  InsightDirection.Up));
            }

            //  Short sell when net imports of petroleum products are decreasing, decreasing demand lowers the
petroleum price and the proxy stock price
            if (currentValue < previousValue)
            {
                insights.Add(Insight.Price(symbol, TimeSpan.FromDays(7),  InsightDirection.Down));
            }
        }

        previousValue = currentValue;
        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            tradableSymbols.Add(security.Symbol);
        }
    }
}
```

Data Point Attributes

The EIA dataset provides `USEnergy` objects, which have the following attributes:

Datasets

ExtractAlpha

ExtractAlpha was founded by Vinesh Jha in 2013 with the goal of providing alternative data for investors. ExtractAlpha's rigorously researched data sets and quantitative stock selection models leverage unique sources and analytical techniques, allowing users to gain an investment edge.

Cross Asset Model

Estimize

Tactical

True Beats

ExtractAlpha

Cross Asset Model

Introduction

The Cross Asset Model by ExtractAlpha provides stock scoring values based on the trading activity in the Options market. Since the Options market has a higher proportion of institutional traders than the Equities market, the Options market is composed of investors who are more informed and information-driven on average. The data covers a dynamic universe of over 3,000 US Equities, starts in July 2005, and is delivered on a daily frequency. This dataset is created by feature engineering on the Options market put-call spread, volatility skewness, and volume.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Cross Asset Model dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

ExtractAlpha was founded by Vinesh Jha in 2013 with the goal of providing alternative data for investors. ExtractAlpha's rigorously researched data sets and quantitative stock selection models leverage unique sources and analytical techniques, allowing users to gain an investment edge.

Getting Started

The following snippet demonstrates how to request data from the Cross Asset Model dataset:

```C#
_symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
_datasetSymbol = AddData<ExtractAlphaCrossAssetModel>(_symbol).Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	July 2005
Asset Coverage	Over 3,000 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Requesting Data

To add Cross Asset Model data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```
C#
public class ExtractAlphaCrossAssetModelDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);

        _symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _datasetSymbol = AddData<ExtractAlphaCrossAssetModel>(_symbol).Symbol;
    }
}
```

Accessing Data

To get the current Cross Asset Model data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var dataPoint = slice[_datasetSymbol];
        Log($"{_datasetSymbol} score at {slice.Time}: {dataPoint.Score}");
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<ExtractAlphaCrossAssetModel>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} score at {slice.Time}: {dataPoint.Score}");
    }
}
```

Historical Data

To get historical Cross Asset Model data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
var history = History<ExtractAlphaCrossAssetModel>(_datasetSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```
C#
RemoveSecurity(_datasetSymbol);
```

If you subscribe to Cross Asset Model data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Cross Asset Model dataset by ExtractAlpha enables you to utilize Options market information to extract alpha. Examples include the following strategies:

- Predicting price and volatility changes in Equities.
- Signaling arbitrage opportunities between Options and underlying assets.
- Using it as a stock selection indicator.

Classic Algorithm Example

The following example algorithm creates a dynamic universe of the 100 most liquid US Equities. Each day, the algorithm forms an equal-weighted dollar-neutral portfolio of the 10 companies most likely to outperform and the 10 companies most likely to underperform.

C#

```
public class ExtractAlphaCrossAssetModelAlgorithm : QCAlgorithm
{
    private DataDictionary<ExtractAlphaCrossAssetModel> _points = new DataDictionary<ExtractAlphaCrossAssetModel>();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        AddUniverse(SelectAssets);
        // Add a Scheduled Event to rebalance the portfolio each day.
        var spy = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
        Schedule.On(DateRules.EveryDay(spy), TimeRules.AfterMarketOpen(spy, 30), Rebalance);
    }

    private IEnumerable<Symbol> SelectAssets(IEnumerable<Fundamental> coarse)
    {
        // Select non-penny stocks with highest dollar volume due to better informed information from more market
activities
        // Only the ones with fundamental data are supported by cross asset model data
        return (from c in coarse
                where c.HasFundamentalData && c.Price > 4
                orderby c.DollarVolume descending
                select c.Symbol).Take(100);
    }

    public override void OnData(Slice slice)
    {
        // Get the current data from the cross asset model.
        var points = slice.Get<ExtractAlphaCrossAssetModel>();
        if (points.Count > 0)
        {
            _points = points;
            //// Demonstrate how to iterate through the data and access its members:
            //foreach(var kvp in points)
            //{
            //    var datasetSymbol = kvp.Key;
            //    var model = kvp.Value;
            //    Quit(
            //        $"{Time} -- " +
            //        $"Asset Symbol: {datasetSymbol.Underlying}; " +
            //        $"Score: {model.Score}"
            //    );
            //}
        }
    }

    public void Rebalance()
    {
        // Long the ones with the highest return estimates based on option trade data
        // Short the lowest return ones
        var sortedByScore = from s in _points.Values
                            where (s.Score != null && Securities[s.Symbol.Underlying].Price != 0)
```

```
                        orderby s.Score
                        select s.Symbol.Underlying;
        var longSymbols = sortedByScore.TakeLast(10);
        var shortSymbols = sortedByScore.Take(10);

        // Liquidate the ones without a strong trading signal
        // Invest equally and dollar-neutral to evenly dissipate capital risk and hedge systematic risk
        var targets = new List<PortfolioTarget>();
        targets.AddRange(longSymbols.Select(symbol => new PortfolioTarget(symbol, 0.05m)));
        targets.AddRange(shortSymbols.Select(symbol => new PortfolioTarget(symbol, -0.05m)));
        SetHoldings(targets, true);
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach(dynamic security in changes.AddedSecurities)
        {
            // Requesting cross asset model data for trading signal generation
            security.CrossAssetModel = AddData<ExtractAlphaCrossAssetModel>(security.Symbol).Symbol;
            // Historical Data
            var history = History(security.CrossAssetModel, 2, Resolution.Daily);
        }
        foreach (dynamic security in changes.RemovedSecurities)
        {
            // Remove the cross asset model data for this asset when it leaves the universe.
            RemoveSecurity(security.CrossAssetModel);
        }
    }
}
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of the 100 most liquid US Equities. Each day, the algorithm forms an equal-weighted dollar-neutral portfolio of the 10 companies most likely to outperform and the 10 companies most likely to underperform.

C#

```
public class ExtractAlphaCrossAssetModelFrameworkAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        AddUniverseSelection(new LiquidEquitiesUniverseSelectionModel());
        // Custom alpha model emits insights based on cross asset model data
        AddAlpha(new ExtractAlphaCrossAssetModelAlphaModel());
        // Invest equally and dollar-neutral to evenly dissipate capital risk and hedge systematic risk
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
        SetExecution(new ImmediateExecutionModel());
    }

    private IEnumerable<Symbol> MyCoarseFilterFunction(IEnumerable<CoarseFundamental> coarse)
    {

        return (from c in coarse
                where c.HasFundamentalData && c.Price > 4
                orderby c.DollarVolume descending
                select c.Symbol).Take(100);
    }
}

public class LiquidEquitiesUniverseSelectionModel : FundamentalUniverseSelectionModel
{
    public override IEnumerable<Symbol> Select(QCAlgorithm algorithm, IEnumerable<Fundamental> fundamentals)
    {
        // Select non-penny stocks with highest dollar volume due to better informed information from more market
activities
        // Only the ones with fundamental data are supported by cross asset model data
        return (from f in fundamentals
                where f.HasFundamentalData && f.Price > 4
                orderby f.DollarVolume descending
                select f.Symbol).Take(100);
    }
}
```

```
public class ExtractAlphaCrossAssetModelAlphaModel: AlphaModel
{
    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        // Get the current data from the cross asset model.
        var points = slice.Get<ExtractAlphaCrossAssetModel>()
            // Drop factors for assets that have no price or score.
            .Where(kvp => algorithm.Securities[kvp.Key.Underlying].Price != 0 && kvp.Value.Score != null);
        // Demonstrate how to iterate through the data and access its members:
        //foreach(var kvp in points)
        //{
        //    var datasetSymbol = kvp.Key;
        //    var model = kvp.Value;
        //    algorithm.Quit(
        //        $"{algorithm.Time} -- " +
        //        $"Asset Symbol: {datasetSymbol.Underlying}; " +
        //        $"Score: {model.Score}"
        //    );
        //}

        // Long the ones with the highest return estimates based on option trade data
        // Short the lowest return ones
        var sortedByScore = points.OrderBy(kvp => kvp.Value.Score).Select(kvp => kvp.Key.Underlying);
        var longSymbols = sortedByScore.TakeLast(10);
        var shortSymbols = sortedByScore.Take(10);

        var insights = new List<Insight>();
        insights.AddRange(longSymbols.Select(symbol => new Insight(symbol, Expiry.EndOfDay, InsightType.Price,
InsightDirection.Up)));
        insights.AddRange(shortSymbols.Select(symbol => new Insight(symbol, Expiry.EndOfDay, InsightType.Price,
InsightDirection.Down)));
        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach(dynamic security in changes.AddedSecurities)
        {
            // Requesting cross asset model data for trading signal generation
            security.CrossAssetModel = algorithm.AddData<ExtractAlphaCrossAssetModel>(security.Symbol).Symbol;
            // Historical Data
            var history = algorithm.History(security.CrossAssetModel, 2, Resolution.Daily);
        }
        foreach (dynamic security in changes.RemovedSecurities)
        {
            // Remove the cross asset model data for this asset when it leaves the universe.
            algorithm.RemoveSecurity(security.CrossAssetModel);
        }
    }
}
```

Data Point Attributes

The Cross Asset Model dataset provides `ExtractAlphaCrossAssetModel` objects, which have the following attributes:

ExtractAlpha

Estimize

Introduction

The Estimize dataset by ExtractAlpha estimates the financials of companies, including EPS, and revenues. The data covers over 2,800 US-listed Equities' EPS/Revenue. The data starts in January 2011 and is updated on a daily frequency. The data is sparse, and it doesn't have new updates every day. This dataset is crowdsourced from a community of 100,000+ contributors via the data provider's web platform.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Estimize dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

ExtractAlpha was founded by Vinesh Jha in 2013 with the goal of providing alternative data for investors. ExtractAlpha's rigorously researched data sets and quantitative stock selection models leverage unique sources and analytical techniques, allowing users to gain an investment edge.

Getting Started

The following snippet demonstrates how to request data from the Estimize dataset:

```C#
_symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
_estimizeConsensusSymbol = AddData<EstimizeConsensus>(_symbol).Symbol;
_estimizeEstimateSymbol = AddData<EstimizeEstimate>(_symbol).Symbol;
_estimizeReleaseSymbol = AddData<EstimizeRelease>(_symbol).Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2011
Asset Coverage	2,800 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Requesting Data

To add Estimize data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```
C#
public class ExtractAlphaEstimizeDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _estimizeConsensusSymbol, _estimizeEstimateSymbol, _estimizeReleaseSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);

        _symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _estimizeConsensusSymbol = AddData<EstimizeConsensus>(_symbol).Symbol;
        _estimizeEstimateSymbol = AddData<EstimizeEstimate>(_symbol).Symbol;
        _estimizeReleaseSymbol = AddData<EstimizeRelease>(_symbol).Symbol;
    }
}
```

Accessing Data

To get the current Estimize data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_estimizeConsensusSymbol))
    {
        var dataPoint = slice[_estimizeConsensusSymbol];
        Log($"{_estimizeConsensusSymbol} mean at {slice.Time}: {dataPoint.Mean}");
    }

    if (slice.ContainsKey(_estimizeEstimateSymbol))
    {
        var dataPoint = slice[_estimizeEstimateSymbol];
        Log($"{_estimizeEstimateSymbol} EPS at {slice.Time}: {dataPoint.Eps}");
    }

    if (slice.ContainsKey(_estimizeReleaseSymbol))
    {
        var dataPoint = slice[_estimizeReleaseSymbol];
        Log($"{_estimizeReleaseSymbol} EPS at {slice.Time}: {dataPoint.Eps}");
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<EstimizeConsensus>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} mean at {slice.Time}: {dataPoint.Mentions}");
    }

    foreach (var kvp in slice.Get<EstimizeEstimate>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} EPS at {slice.Time}: {dataPoint.Eps}");
    }

    foreach (var kvp in slice.Get<EstimizeRelease>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} EPS at {slice.Time}: {dataPoint.Eps}");
    }
}
```

Historical Data

To get historical Estimize data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
// Dataset objects
var concensusHistory = History<EstimizeConsensus>(_estimizeConsensusSymbol, 100, Resolution.Daily);
var estimateHistory = History<EstimizeEstimate>(_estimizeEstimateSymbol, 100, Resolution.Daily);
var releaseHistory = History<EstimizeRelease>(_estimizeReleaseSymbol, 100, Resolution.Daily);

// Slice objects
var history = History(new[]{_estimizeConsensusSymbol,
                            _estimizeEstimateSymbol,
                            _estimizeReleaseSymbol}, 10, Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_estimizeConsensusSymbol);
RemoveSecurity(_estimizeEstimateSymbol);
RemoveSecurity(_estimizeReleaseSymbol);
```

If you subscribe to Estimize data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Estimize dataset enables you to estimate the financial data of a company more accurately for alpha. Examples include the following use cases:

- Fundamental estimates for ML regression/classification models
- Arbitrage/Sentiment trading on market “surprise” from ordinary expectations based on the better expectation by the dataset

- Using industry-specific KPIs to predict the returns of individual sectors

Classic Algorithm Example

The following example algorithm creates a dynamic universe of the 250 most liquid US Equities. Each morning, the algorithm forms an equal-weighted with the companies that have a new EPS estimate.

C#

```
public class ExtractAlphaEstimizeAlgorithm : QCAlgorithm
{
    // Add a dictionary to hold the latest Consensus data since this dataset is sparse.
    private Dictionary<Symbol, EstimizeConsensus> _consensusBySymbol = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        // Seed securities with their last known prices to avoid stale-price trade errors
        Settings.SeedInitialPrices = true;
        // Add a universe of liquid US Equities.
        AddUniverse(SelectAssets);
        // Add a Scheduled Event to rebalance the portfolio each day.
        var spy = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
        Schedule.On(
            DateRules.EveryDay(spy),
            TimeRules.AfterMarketOpen(spy, 1),
            Rebalance
        );
    }

    private IEnumerable<Symbol> SelectAssets(IEnumerable<Fundamental> fundamentals)
    {
        // Select the non-penny stocks with the highest dollar volume,
        // since they have more stable price (lower risk) and more
        // informed insights from high market activities
        return fundamentals
            .Where(x => x.HasFundamentalData && x.Price > 4)
            .OrderBy(x => x.DollarVolume)
            .TakeLast(250)
            .Select(x => x.Symbol);
    }

    public override void OnData(Slice slice)
    {
        // Record the latest Consensus data.
        var consensus = slice.Get<EstimizeConsensus>();
        if (consensus.IsNullOrEmpty()) return;
        foreach (var kvp in consensus)
        {
            _consensusBySymbol[kvp.Key.Underlying] = kvp.Value;
        }
    }

    private void Rebalance()
    {
        if (_consensusBySymbol.Count == 0) return;
        // If there has been new Consensus data released since the previous
        // market open, form an equal-weighted portfolio of all the assets
        // that have new Consensus data.
        var symbols = _consensusBySymbol.Keys.Where(symbol => Securities[symbol].Price > 0);
        var targets = symbols.Select(symbol => new PortfolioTarget(symbol, 1m/symbols.Count()));
        SetHoldings(targets.ToList(), true);
        _consensusBySymbol.Clear();
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach (dynamic security in changes.AddedSecurities)
        {
            // Requesting data for trading signal generation
            security.Consensus = AddData<EstimizeConsensus>(security.Symbol).Symbol;
            // Historical data
            var history = History<EstimizeConsensus>(security.Consensus, 100, Resolution.Daily);
        }

        foreach (dynamic security in changes.RemovedSecurities)
        {
            // Remove the Consensus data for this asset when it leaves the universe.
            RemoveSecurity(security.Consensus);
        }
    }
}
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of the 250 most liquid US Equities. Each morning, the algorithm emits insights for the companies that have a new EPS estimate.

C#

```
public class ExtractAlphaEstimizeFrameworkAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        AddUniverseSelection(new LiquidEquitiesUniverseSelectionModel());
        // Custom alpha model that emit signal according to Estimize data
        AddAlpha(new ExtractAlphaEstimizeAlphaModel());
        // Invest equally to evenly dissipate capital concentration risk
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
        SetExecution(new ImmediateExecutionModel());
    }
}

public class LiquidEquitiesUniverseSelectionModel : FundamentalUniverseSelectionModel
{
    public override IEnumerable<Symbol> Select(QCAlgorithm algorithm, IEnumerable<Fundamental> fundamentals)
    {
        // Select the non-penny stocks with the highest dollar volume,
        // since they have more stable price (lower risk) and more
        // informed insights from high market activities
        return fundamentals
            .Where(x => x.HasFundamentalData && x.Price > 4)
            .OrderBy(x => x.DollarVolume)
            .TakeLast(250)
            .Select(x => x.Symbol);
    }
}

public class ExtractAlphaEstimizeAlphaModel: AlphaModel
{
    // Add a dictionary to hold the latest Consensus data since this dataset is sparse.
    private Dictionary<Symbol, EstimizeConsensus> _consensusBySymbol = new();
    private int _day = -1;

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = new List<Insight>();
        // Record the latest Consensus data.
        var consensus = slice.Get<EstimizeConsensus>();
        if (!consensus.IsNullOrEmpty())
        {
            foreach (var kvp in consensus)
            {
                _consensusBySymbol[kvp.Key.Underlying] = kvp.Value;
            }
        }
        // Wait for the market open to rebalance
        if (_day == algorithm.Time.Day || slice.Bars.Count == 0)
        {
            return insights;
        }
        _day = algorithm.Time.Day;
        // If there has been new Consensus data released since the previous
        // market open, emit insights for all the assets that have new Consensus data.
        if (_consensusBySymbol.Count == 0)
        {
            return insights;
        }
        var symbols = _consensusBySymbol.Keys.Where(symbol => algorithm.Securities[symbol].Price != 0);
        insights.AddRange(symbols.Select(symbol => Insight.Price(symbol, TimeSpan.FromDays(30),
InsightDirection.Up)).ToList());
        _consensusBySymbol.Clear();
        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach (dynamic security in changes.AddedSecurities)
        {
            // Requesting data for trading signal generation
            security.Consensus = algorithm.AddData<EstimizeConsensus>(security.Symbol).Symbol;
            // Historical data
```

```
            var history = algorithm.History<EstimizeConsensus>(security.Consensus, 100, Resolution.Daily);
        }

        foreach (dynamic security in changes.RemovedSecurities)
        {
            // Remove the Consensus data for this asset when it leaves the universe.
            algorithm.RemoveSecurity(security.Consensus);
        }
    }
}
```

Data Point Attributes

The Estimize dataset provides `EstimizeConsensus` , `EstimizeEstimate` , and `EstimizeRelease` objects.

EstimizeConsensus Attributes

`EstimizeConsensus` objects have the following attributes:

EstimizeEstimate Attributes

`EstimizeEstimate` objects have the following attributes:

EstimizeRelease Attributes

`EstimizeRelease` objects have the following attributes:

ExtractAlpha

Tactical

Introduction

The Tactical dataset by ExtractAlpha is a stock scoring algorithm that captures the technical dynamics of individual US Equities over one to ten trading day horizons. It can assist a longer-horizon investor in timing their entry or exit points or be used in combination with existing systematic or qualitative strategies with similar holding periods.

The data covers a dynamic universe of around 4,700 US Equities per day on average, starts in January 2000, and is delivered on a daily frequency. The Tactical dataset expands upon simple reversal, liquidity, and seasonality factors to identify stocks that are likely to trend or reverse.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Tactical dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

ExtractAlpha was founded by Vinesh Jha in 2013 with the goal of providing alternative data for investors. ExtractAlpha's rigorously researched data sets and quantitative stock selection models leverage unique sources and analytical techniques, allowing users to gain an investment edge.

Getting Started

The following snippet demonstrates how to request data from the Tactical dataset:

```
C#
_symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
_datasetSymbol = AddData<ExtractAlphaTacticalModel>(_symbol).Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2000
Asset Coverage	5,000 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Requesting Data

To add Tactical data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the

data later in your algorithm.

```
C#
public class ExtractAlphaTacticalModelDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);

        _symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _datasetSymbol = AddData<ExtractAlphaTacticalModel>(_symbol).Symbol;
    }
}
```

Accessing Data

To get the current Tactical data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var dataPoint = slice[_datasetSymbol];
        Log($"{_datasetSymbol} score at {slice.Time}: {dataPoint.Score}");
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<ExtractAlphaTacticalModel>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} score at {slice.Time}: {dataPoint.Score}");
    }
}
```

Historical Data

To get historical Tactical data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
var history = History<ExtractAlphaTacticalModel>(_datasetSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```
RemoveSecurity(_datasetSymbol);
```

If you subscribe to Tactical data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Tactical dataset enables you to gain insight into short-term stock dynamics for trading. Examples include the following strategies:

- Optimizing entry and exit times in a portfolio construction model.
- Using the raw factor values as technical indicators.
- Inputting the data into machine learning classifier models as trend/reversal labels.

Classic Algorithm Example

The following example algorithm creates a dynamic universe of the 100 most liquid US Equities. Each day, it then forms an equal-weighted dollar-neutral portfolio of the 10 companies that are most likely to outperform and the 10 that are most likely to underperform.

C#

```
public class ExtractAlphaTacticalModelAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        AddUniverse(SelectAssets);
    }

    private IEnumerable<Symbol> SelectAssets(IEnumerable<Fundamental> coarse)
    {
        // Select non-penny stocks with highest dollar volume due to better informed information from more market
activities
        // Only the ones with fundamental data are supported by tactical data
        return (from c in coarse
                where c.HasFundamentalData && c.Price > 4
                orderby c.DollarVolume descending
                select c.Symbol).Take(100);
    }

    public override void OnData(Slice slice)
    {
        // Get the current data from the tactical model.
        var points = slice.Get<ExtractAlphaTacticalModel>()
            // Drop factors for assets that have no price.
            .Where(kvp => Securities[kvp.Key.Underlying].Price != 0);
        //// Demonstrate how to iterate through the data and access its members:
        //foreach(var kvp in points)
        //{
        //    var datasetSymbol = kvp.Key;
        //    var model = kvp.Value;
        //    Quit(
        //        $"{Time} -- " +
        //        $"Asset Symbol: {datasetSymbol.Underlying}; " +
        //        $"Score: {model.Score}"
        //    );
        //}

        // Only rebalance when there are new tactical factors.
        if (points.Count() == 0)
        {
            return;
        }

        // Long the ones with the highest return estimates riding from tactical strategies
        // Short the lowest that predicted stock price goes down
        var sortedByScore =  points.OrderBy(kvp => kvp.Value.Score)
            .Select(kvp => kvp.Key.Underlying);
        var longSymbols = sortedByScore.TakeLast(10);
        var shortSymbols = sortedByScore.Take(10);

        // Invest equally and dollar-neutral to evenly dissipate capital risk and hedge systematic risk
        var targets = new List<PortfolioTarget>();
        targets.AddRange(longSymbols.Select(symbol => new PortfolioTarget(symbol, 0.05m)));
        targets.AddRange(shortSymbols.Select(symbol => new PortfolioTarget(symbol, -0.05m)));

        SetHoldings(targets, true);
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach(dynamic security in changes.AddedSecurities)
        {
            // Requesting tactical data for trading signal generation
            security.TacticalModel = AddData<ExtractAlphaTacticalModel>(security.Symbol).Symbol;
            // Historical Data
            var history = History(security.TacticalModel, 2, Resolution.Daily);
        }
        foreach (dynamic security in changes.RemovedSecurities)
        {
            // Remove the tactical data for this asset when it leaves the universe.
            RemoveSecurity(security.TacticalModel);
        }
    }
}
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of the 100 most liquid US Equities. Each day, it then forms an equal-weighted dollar-neutral portfolio of the 10 companies that are most likely to outperform and the 10 that are most likely to underperform.

C#

```
public class ExtractAlphaTacticalModelFrameworkAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        AddUniverseSelection(new LiquidEquitiesUniverseSelectionModel());
        // Custom alpha model to generate trading signal based on tactical data
        AddAlpha(new ExtractAlphaTacticalModelAlphaModel());
        // Invest equally and dollar-neutral to evenly dissipate capital risk and hedge systematic risk
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
        SetExecution(new ImmediateExecutionModel());
    }
}

public class LiquidEquitiesUniverseSelectionModel : FundamentalUniverseSelectionModel
{
    public override IEnumerable<Symbol> Select(QCAlgorithm algorithm, IEnumerable<Fundamental> fundamentals)
    {
        // Select non-penny stocks with highest dollar volume due to better informed information from more market
activities
        // Only the ones with fundamental data are supported by tactical data
        return (from f in fundamentals
                where f.HasFundamentalData && f.Price > 4
                orderby f.DollarVolume descending
                select f.Symbol).Take(100);
    }
}

public class ExtractAlphaTacticalModelAlphaModel: AlphaModel
{
    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        // Get the current data from the tactical model.
        var points = slice.Get<ExtractAlphaTacticalModel>()
            // Drop factors for assets that have no price or score.
            .Where(kvp => algorithm.Securities[kvp.Key.Underlying].Price != 0);
        //// Demonstrate how to iterate through the data and access its members:
        //foreach(var kvp in points)
        //{
        //    var datasetSymbol = kvp.Key;
        //    var model = kvp.Value;
        //    algorithm.Quit(
        //        $"{algorithm.Time} -- " +
        //        $"Asset Symbol: {datasetSymbol.Underlying}; " +
        //        $"Score: {model.Score}"
        //    );
        //}

        // Long the ones with the highest return estimates riding from tactical strategies
        // Short the lowest that predicted stock price goes down
        var sortedByScore =  points.OrderBy(kvp => kvp.Value.Score)
            .Select(kvp => kvp.Key.Underlying);
        var longSymbols = sortedByScore.TakeLast(10);
        var shortSymbols = sortedByScore.Take(10);

        var insights = new List<Insight>();
        insights.AddRange(longSymbols.Select(symbol => new Insight(symbol, Expiry.EndOfDay, InsightType.Price,
InsightDirection.Up)));
        insights.AddRange(shortSymbols.Select(symbol => new Insight(symbol, Expiry.EndOfDay, InsightType.Price,
InsightDirection.Down)));
        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach(dynamic security in changes.AddedSecurities)
        {
            // Requesting tactical data for trading signal generation
            security.TacticalModel = algorithm.AddData<ExtractAlphaTacticalModel>(security.Symbol).Symbol;
```

```
            // Historical Data
            var history = algorithm.History(security.TacticalModel, 2, Resolution.Daily);
        }
        foreach (dynamic security in changes.RemovedSecurities)
        {
            // Remove the tactical model data for this asset when it leaves the universe.
            algorithm.RemoveSecurity(security.TacticalModel);
        }
    }
}
```

Data Point Attributes

The Tactical dataset provides `ExtractAlphaTacticalModel` objects, which have the following attributes:

ExtractAlpha

True Beats

Introduction

The True Beats dataset by ExtractAlpha provides quantitative predictions of EPS and Revenues for US Equities. The data covers a dynamic universe of around 4,000-5,000 US-listed Equities on a daily average. The data starts in January 2000 and is delivered on a daily frequency. This dataset is created by incorporating the opinions of expert analysts, historical earnings, revenue trends for the company and its peers, and metrics on company earnings management.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the True Beats dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

ExtractAlpha was founded by Vinesh Jha in 2013 with the goal of providing alternative data for investors. ExtractAlpha's rigorously researched data sets and quantitative stock selection models leverage unique sources and analytical techniques, allowing users to gain an investment edge.

Getting Started

The following snippet demonstrates how to request data from the True Beats dataset:

```C#
_symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
_datasetSymbol = AddData<ExtractAlphaTrueBeats>(_symbol).Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2000
Asset Coverage	Over 5,000 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Requesting Data

To add True Beats data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```
C#
public class ExtractAlphaTrueBeatsDataAlgorithm: QCAlgorithm
{
    private Symbol _symbol, _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);

        _symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _datasetSymbol = AddData<ExtractAlphaTrueBeats>(_symbol).Symbol;
    }
}
```

Accessing Data

To get the current True Beats data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var dataPoint = slice[_datasetSymbol];
        Log($"{_datasetSymbol} true beat at {slice.Time}: {dataPoint.TrueBeat}");
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<ExtractAlphaTrueBeats>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} true beat at {slice.Time}: {dataPoint.TrueBeat}");
    }
}
```

Historical Data

To get historical True Beats data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
var history = History<ExtractAlphaTrueBeats>(_datasetSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```
C#
RemoveSecurity(_datasetSymbol);
```

If you subscribe to True Beats data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The True Beats dataset enables you to predict EPS and revenue of US-listed Equities for trading. Examples include the following strategies:

- Finding surprise in EPS or revenue for sentiment/arbitrage trading
- Stock or sector selections based on EPS or revenue predictions
- Calculate expected return by valuation models based on EPS or revenue predictions (e.g. Black-Litterman)

Classic Algorithm Example

The following example algorithm creates a dynamic universe of the 100 most liquid US Equities. Each day, it then forms an equal-weighted dollar-neutral portfolio of the 10 best and 10 worst companies based on their predicted earnings per share.

```C#
public class ExtractAlphaTrueBeatsDataAlgorithm : QCAlgorithm
{
    private DataDictionary<ExtractAlphaTrueBeats> _trueBeats;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 30);
        SetCash(100000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Add a universe that runs at the start of each month.
        var spy = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
        var dateRule = DateRules.MonthStart(spy);
        UniverseSettings.Schedule.On(dateRule);
        AddUniverse(SelectAssets);
        // Add a Scheduled Event to rebalance the portfolio at the start of each month.
        Schedule.On(dateRule, TimeRules.BeforeMarketClose(spy, 30), Rebalance);
    }

    private IEnumerable<Symbol> SelectAssets(IEnumerable<Fundamental> coarse)
    {
        // Select the stocks with highest dollar volume due to better informed information from more market
activities
        // Only the ones with fundamental data are supported by true beats data
        return (from c in coarse
                where c.HasFundamentalData
                orderby c.DollarVolume descending
                select c.Symbol).Take(100);
    }

    public override void OnData(Slice slice)
    {
        // Get the current data from the true beats dataset.
        var points = slice.Get<ExtractAlphaTrueBeats>();
        if (!points.IsNullOrEmpty())
        {
            _trueBeats = points;
        }
    }

    public void Rebalance()
    {
        var epsPredictionBySymbol = new Dictionary<Symbol, decimal>();
        foreach (var kvp in _trueBeats)
        {
            var datasetSymbol = kvp.Key;
            // Select the subset of TrueBeat objects that have the EPS earnings metric,
            // a value for the fiscal year, and a value for the fiscal quarter.
            var trueBeats = kvp.Value
                .Select(tb => (ExtractAlphaTrueBeat)tb)
                .Where(tb =>
                    tb.EarningsMetric == ExtractAlphaTrueBeatEarningsMetric.EPS &&
                    tb.FiscalPeriod.FiscalQuarter != null
                )
```

```
                .OrderBy(tb => tb.FiscalPeriod.FiscalYear)
                .ThenBy(tb => tb.FiscalPeriod.FiscalQuarter)
                .ToList();
            if (trueBeats.Count == 0)
                continue;
            // Select the TrueBeat object of the closest fiscal quarter.
            epsPredictionBySymbol[datasetSymbol.Underlying] = trueBeats.First().TrueBeat;
        }

        // Long the ones with the highest earning and revenue estimates due to fundamental factor may bring stock
price up
        // Short the lowest that predicted to bring stock price down
        // Invest equally and dollar-neutral to evenly dissipate capital risk and hedge systematic risk
        var sortedByEpsPrediction = epsPredictionBySymbol.OrderBy(kvp => kvp.Value).ToList();
        var shortTargets = sortedByEpsPrediction.Take(10).Select(kvp => new PortfolioTarget(kvp.Key, -0.05m));
        var longTargets = sortedByEpsPrediction.TakeLast(10).Select(kvp => new PortfolioTarget(kvp.Key, 0.05m));
        SetHoldings(longTargets.Concat(shortTargets).ToList(), true);
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach(dynamic security in changes.AddedSecurities)
        {
            // Requesting true beats data for trading signal generation
            security.TrueBeats = AddData<ExtractAlphaTrueBeats>(security.Symbol).Symbol;
            // Historical Data
            var history = History(security.TrueBeats, 10, Resolution.Daily);
        }
        foreach (dynamic security in changes.RemovedSecurities)
        {
            // Remove the true beats data for this asset when it leaves the universe.
            RemoveSecurity(security.TrueBeats);
        }
    }
}
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of the 100 most liquid US Equities. Each day, it then forms an equal-weighted dollar-neutral portfolio of the 10 best and 10 worst companies based on their predicted earnings per share.

C#

```
public class ExtractAlphaTrueBeatsDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Add a universe that runs at the start of each month.
        var spy = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
        var dateRule = DateRules.MonthStart(spy);
        UniverseSettings.Schedule.On(dateRule);
        AddUniverseSelection(new LiquidEquitiesUniverseSelectionModel());
        // Custom alpha model to trade based on true beats data signals
        AddAlpha(new ExtractAlphaTrueBeatsAlphaModel());
        // Invest equally and dollar-neutral to evenly dissipate capital risk and hedge systematic risk
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
        SetExecution(new ImmediateExecutionModel());
    }
}

public class LiquidEquitiesUniverseSelectionModel : FundamentalUniverseSelectionModel
{
    public override IEnumerable<Symbol> Select(QCAlgorithm algorithm, IEnumerable<Fundamental> fundamentals)
    {
        // Select non-penny stocks with highest dollar volume due to better informed information from more market
activities
        // Only the ones with fundamental data are supported by true beats data
        return (from f in fundamentals
                where f.HasFundamentalData
                orderby f.DollarVolume descending
                select f.Symbol).Take(100);
    }
```

```
}

public class ExtractAlphaTrueBeatsAlphaModel: AlphaModel
{
    private int _month = -1;
    private List<Insight> _insights = new();

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        // Rebalance at the start of each month.
        if (_month == algorithm.Time.Month)
        {
            return new List<Insight>();
        }
        // Trade only based on the updated true beats data
        var points = slice.Get<ExtractAlphaTrueBeats>();
        if (points.IsNullOrEmpty())
        {
            return new List<Insight>();
        }
        _month = algorithm.Time.Month;

        // Get the EPS prediction of each asset.
        var epsPredictionBySymbol = new Dictionary<Symbol, decimal>();
        foreach (var kvp in points)
        {
            var datasetSymbol = kvp.Key;
            // Select the subset of TrueBeat objects that have the EPS earnings metric,
            // a value for the fiscal year, and a value for the fiscal quarter.
            var trueBeats = kvp.Value
                .Select(tb => (ExtractAlphaTrueBeat)tb)
                .Where(tb =>
                    tb.EarningsMetric == ExtractAlphaTrueBeatEarningsMetric.EPS &&
                    tb.FiscalPeriod.FiscalQuarter != null
                )
                .OrderBy(tb => tb.FiscalPeriod.FiscalYear)
                .ThenBy(tb => tb.FiscalPeriod.FiscalQuarter)
                .ToList();
            if (trueBeats.Count == 0)
                continue;
            // Select the TrueBeat object of the closest fiscal quarter.
            epsPredictionBySymbol[datasetSymbol.Underlying] = trueBeats.First().TrueBeat;
        }

        // Cancel the previous insights.
        algorithm.Insights.Cancel(_insights);

        // Emit the new insights.
        // Long the ones with the highest earning and revenue estimates due to fundamental factor may bring stock
price up
        // Short the lowest that predicted to bring stock price down
        // Invest equally and dollar-neutral to evenly dissipate capital risk and hedge systematic risk
        _insights.Clear();
        var sortedByEpsPrediction = epsPredictionBySymbol.OrderBy(kvp => kvp.Value).ToList();
        _insights.AddRange(sortedByEpsPrediction.TakeLast(10).Select(kvp => Insight.Price(kvp.Key,
TimeSpan.FromDays(35), InsightDirection.Up)));
        _insights.AddRange(sortedByEpsPrediction.Take(10).Select(kvp => Insight.Price(kvp.Key, TimeSpan.FromDays(35),
InsightDirection.Down)));
        return _insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach(dynamic security in changes.AddedSecurities)
        {
            // Requesting true beats data for trading signal generation
            security.TrueBeats = algorithm.AddData<ExtractAlphaTrueBeats>(security.Symbol).Symbol;
            // Historical Data
            var history = algorithm.History(security.TrueBeats, 10, Resolution.Daily);
        }
        foreach (dynamic security in changes.RemovedSecurities)
        {
            // Remove the true beats data for this asset when it leaves the universe.
            algorithm.RemoveSecurity(security.TrueBeats);
        }
    }
}
```

Data Point Attributes

The True Beats dataset provides `ExtractAlphaTrueBeats` and `ExtractAlphaTrueBeat` objects.

ExtractAlphaTrueBeats

`ExtractAlphaTrueBeats` objects have the following attributes:

ExtractAlphaTrueBeat

`ExtractAlphaTrueBeat` objects have the following attributes:

Datasets

FRED

The Research Division of the Federal Reserve bank of St. Louis, MO expands the frontier of economic knowledge by producing high-quality original research in the areas of macroeconomics, money and banking, and applied microeconomics. They contribute to monetary policy discussions by advising the Bank president on a range of topics, especially in preparation for Federal Open Market Committee (FOMC) meetings. The Research Division is in the top 1% of all economics research departments worldwide.

US Federal Reserve (FRED)

FRED

US Federal Reserve (FRED)

Introduction

The Federal Reserve Economic Data (FRED) by the Research Division of the Federal Reserve bank of St. Louis, MO provides various time series relating to macro-economic data. The data covers 560 datasets, starts in January 1999, and is delivered on a daily frequency. The data is created by aggregating daily updates from more than 85 public and proprietary sources.

For more information about the US Federal Reserve (FRED) dataset, including CLI commands and pricing, see the dataset listing
.

About the Provider

The Research Division of the Federal Reserve bank of St. Louis, MO expands the frontier of economic knowledge by producing high-quality original research in the areas of macroeconomics, money and banking, and applied microeconomics. They contribute to monetary policy discussions by advising the Bank president on a range of topics, especially in preparation for Federal Open Market Committee (FOMC) meetings. The Research Division is in the top 1% of all economics research departments worldwide.

Getting Started

The following snippet demonstrates how to request data from the FRED dataset:

```C#
_datasetSymbol = AddData<Fred>("USARECDM", Resolution.Daily).Symbol;
```

QuantConnect provides helpers to find some of the available datasets from FRED:

```C#
_datasetSymbol = AddData<Fred>(Fred.OECDRecessionIndicators.UnitedStatesFromPeakThroughTheTrough,
Resolution.Daily).Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 1990
Asset Coverage	560 Datasets
Data Density	Sparse
Resolution	Daily
Timezone	New York

Requesting Data

To add FRED data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```
C#
public class FredAlternativeDataAlgorithm : QCAlgorithm
{
    private Symbol _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2003, 1, 1);
        SetEndDate(2019, 10, 11);
        SetCash(100000);

        _datasetSymbol = AddData<Fred>(Fred.OECDRecessionIndicators.UnitedStatesFromPeakThroughTheTrough,
Resolution.Daily).Symbol;
    }
}
```

Accessing Data

To get the current FRED data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var dataPoint = slice[_datasetSymbol];
        Log($"{_datasetSymbol} value at {slice.Time}: {dataPoint.Value}");
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<Fred>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} value at {slice.Time}: {dataPoint.Value}");
    }
}
```

Historical Data

To get historical FRED data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
var history = History<Fred>(_datasetSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove your subscription to FRED data, call the `RemoveSecurity` method.

```
RemoveSecurity(_datasetSymbol);
```
C#

Supported Datasets

The following lists show the accessor code you need to add each FRED dataset to your algorithm. This list does `not` include all fields from the FRED database. Please refer to FRED website for additional fields.

CBOE

- `Fred.CBOE.VIXOnGoogle` : CBOE Equity VIX on Google (in Index) (`VXGOGCLS`)
- `Fred.CBOE.VXD` : CBOE DJIA Volatility Index (in Index) (`VXDCLS`)
- `Fred.CBOE.VIXOnGoldmanSachs` : CBOE Equity VIX on Goldman Sachs (in Index) (`VXGSCLS`)
- `Fred.CBOE.VIXOnIBM` : CBOE Equity VIX on IBM (in Index) (`VXIBMCLS`)
- `Fred.CBOE.VIXOnAmazon` : CBOE Equity VIX on Amazon (in Index) (`VXAZNCLS`)
- `Fred.CBOE.VXO` : CBOE S&P 100 Volatility Index: VXO (in Index) (`VXOCLS`)
- `Fred.CBOE.VXN` : CBOE NASDAQ 100 Volatility Index (in Index) (`VXNCLS`)
- `Fred.CBOE.TenYearTreasuryNoteVolatilityFutures` : CBOE 10-Year Treasury Note Volatility Futures (in Index) (`VXTYN`)
- `Fred.CBOE.RVX` : CBOE Russell 2000 Volatility Index (in Index) (`RVXCLS`)
- `Fred.CBOE.SP500ThreeMonthVolatilityIndex` : CBOE S&P 500 3-Month Volatility Index (in Index) (`VXVCLS`)
- `Fred.CBOE.VIXOnApple` : CBOE Equity VIX on Apple (in Index) (`VXAPLCLS`)
- `Fred.CBOE.GoldMinersETFVolatilityIndex` : CBOE Gold Miners ETF Volatility Index (in Index) (`VXGDXCLS`)
- `Fred.CBOE.ChinaETFVolatilityIndex` : CBOE China ETF Volatility Index (in Index) (`VXFXICLS`)
- `Fred.CBOE.BrazilETFVolatilityIndex` : CBOE Brazil ETF Volatility Index (in Index) (`VXEWZCLS`)
- `Fred.CBOE.EmergingMarketsETFVolatilityIndex` : CBOE Emerging Markets ETF Volatility Index (in Index) (`VXEEMCLS`)
- `Fred.CBOE.EuroCurrencyETFVolatilityIndex` : CBOE EuroCurrency ETF Volatility Index (in Index) (`EVZCLS`)
- `Fred.CBOE.GoldETFVolatilityIndex` : CBOE Gold ETF Volatility Index (in Index) (`GVZCLS`)
- `Fred.CBOE.CrudeOilETFVolatilityIndex` : CBOE Crude Oil ETF Volatility Index (in Index) (`OVXCLS`)
- `Fred.CBOE.SilverETFVolatilityIndex` : CBOE Silver ETF Volatility Index (in Index) (`VXSLVCLS`)
- `Fred.CBOE.EnergySectorETFVolatilityIndex` : CBOE Energy Sector ETF Volatility Index (in Index) (`VXXLECLS`)
- `Fred.CBOE.VIX` : CBOE Volatility Index: VIX (in Index) (`VIXCLS`)

Central Bank Interventions

- `Fred.CentralBankInterventions.JapaneseBankPurchasesOfDmEuroAgainstJpy` : Japan Intervention: Japanese Bank purchases of DM/Euro against JPY (in 100 Million Yen) (`JPINTDDMEJPY`)
- `Fred.CentralBankInterventions.JapaneseBankPurchasesOfUsdAgainstDm` : Japan Intervention: Japanese Bank purchases of USD against DM (in 100 Million Yen) (`JPINTDEXR`)
- `Fred.CentralBankInterventions.JapaneseBankPurchasesOfUsdAgainstRupiah` : Japan Intervention: Japanese Bank purchases of USD against Rupiah (in 100 Million Yen) (`JPINTDUSDRP`)
- `Fred.CentralBankInterventions.USInterventionInMarketTransactionsInTheJpyUsd` : U.S. Intervention: in Market Transactions in the JPY/USD (Millions of USD) (in Millions of USD) (`USINTDMRKTJPY`)
- `Fred.CentralBankInterventions.USInterventionWithCustomerTransactionsInOtherCurrencies` : U.S. Intervention: With-Customer Transactions in Other Currencies (Millions of USD) (in Millions of USD) (`USINTDCSOTH`)
- `Fred.CentralBankInterventions.USInterventionWithCustomerTransactionsInTheJpyUsd` : U.S. Intervention: With-Customer Transactions in the JPY/USD (Millions of USD) (in Millions of USD) (`USINTDCSJPY`)
- `Fred.CentralBankInterventions.USInterventionWithCustomerTransactionsInTheDemUsdEuro` : U.S. Intervention: With-Customer Transactions in the DEM/USD (Euro since 1999) (Millions of USD) (in Millions of USD) (`USINTDCSDM`)

- Fred.CentralBankInterventions.USInterventionInMarketTransactionsInOtherCurrencies : U.S. Intervention: in Market Transactions in Other Currencies (Millions of USD) (in Millions of USD) (USINTDMRKTOTH)
- Fred.CentralBankInterventions.CentralBankOfTurkeyPurchasesOfUsd : Turkish Intervention: Central Bank of Turkey Purchases of USD (Millions of USD) (in Millions of USD) (TRINTDEXR)
- Fred.CentralBankInterventions.JapaneseBankPurchasesOfUsdAgainstJpy : Japan Intervention: Japanese Bank purchases of USD against JPY (in 100 Million Yen) (JPINTDUSDJPY)
- Fred.CentralBankInterventions.USInterventionInMarketTransactionsInTheDemUsdEuro : U.S. Intervention: in Market Transactions in the DEM/USD (Euro since 1999) (Millions of USD) (in Millions of USD) (USINTDMRKTDM)
- Fred.CentralBankInterventions.SwissNationalBankPurchasesOfDemAgainstChfMillionsOfDem : Swiss Intervention: Swiss National Bank Purchases of DEM against CHF (Millions of DEM) (in Millions of DEM) (CHINTDCHFDM)
- Fred.CentralBankInterventions.SwissNationalBankPurchasesOfUsdAgainstDem : Swiss Intervention: Swiss National Bank Purchases of USD against DEM (Millions of USD) (in Millions of USD) (CHINTDUSDDM)
- Fred.CentralBankInterventions.SwissNationalBankPurchasesOfUsdAgainstJpy : Swiss Intervention: Swiss National Bank Purchases of USD against JPY (Millions of USD) (in Millions of USD) (CHINTDUSDJPY)
- Fred.CentralBankInterventions.SwissNationalBankPurchasesOfUsdAgainstChf : Swiss Intervention: Swiss National Bank Purchases of USD against CHF (Millions of USD) (in Millions of USD) (CHINTDCHFUSD)
- Fred.CentralBankInterventions.BancoDeMexicoPurchaseOnTheUsd : Mexican Intervention: Banco de Mexico Purchase on the USD (in Millions of USD) (MEXINTDUSD)

Commercial Paper

- Fred.CommercialPaper.ThreeMonthAANonfinancialCommercialPaperRate : 3-Month AA Nonfinancial Commercial Paper Rate (in Percent) (DCPN3M)
- Fred.CommercialPaper.OneMonthAANonfinancialCommercialPaperRate : 1-Month AA Nonfinancial Commercial Paper Rate (in Percent) (DCPN30)
- Fred.CommercialPaper.TwoMonthAANonfinancialCommercialPaperRate : 2-Month AA Nonfinancial Commercial Paper Rate (in Percent) (DCPN2M)
- Fred.CommercialPaper.ThreeMonthAAFinancialCommercialPaperRate : 3-Month AA Financial Commercial Paper Rate (in Percent) (DCPF3M)
- Fred.CommercialPaper.TwoMonthAAFinancialCommercialPaperRate : 2-Month AA Financial Commercial Paper Rate (in Percent) (DCPF2M)
- Fred.CommercialPaper.OneMonthAAFinancialCommercialPaperRate : 1-Month AA Financial Commercial Paper Rate (in Percent) (DCPF1M)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween1and4DaysUsedForA2P2Nonfinancial : Number of Issues, with a Maturity Between 1 and 4 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Number) (NONFIN14A2P2VOL)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween5and9DaysUsedForA2P2Nonfinancial : Number of Issues, with a Maturity Between 5 and 9 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Number) (NONFIN59A2P2VOL)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween5and9DaysUsedForA2P2Nonfinancial : Total Value of Issues, with a Maturity Between 5 and 9 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Millions of Dollars) (NONFIN59A2P2AMT)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween41and80DaysUsedForAANonfinancial : Number of Issues, with a Maturity Between 41 and 80 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Number) (NONFIN4180AAVOL)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityGreaterThan80DaysUsedForAAAssetBacked : Total Value of

Issues, with a Maturity Greater Than 80 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Millions of Dollars) (ABGT80AAAMT)

- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween41and80DaysUsedForAANonfinancial : Total Value of Issues, with a Maturity Between 41 and 80 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Millions of Dollars) (NONFIN4180AAAMT)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween41and80DaysUsedForA2P2Nonfinancial : Number of Issues, with a Maturity Between 41 and 80 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Number) (NONFIN4180A2P2VOL)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween41and80DaysUsedForA2P2Nonfinancial : Total Value of Issues, with a Maturity Between 41 and 80 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Millions of Dollars) (NONFIN4180A2P2AMT)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween21and40DaysUsedForAANonfinancial : Number of Issues, with a Maturity Between 21 and 40 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Number) (NONFIN2140AAVOL)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween21and40DaysUsedForAANonfinancial : Total Value of Issues, with a Maturity Between 21 and 40 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Millions of Dollars) (NONFIN2140AAAMT)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween21and40DaysUsedForA2P2Nonfinancial : Number of Issues, with a Maturity Between 21 and 40 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Number) (NONFIN2140A2P2VOL)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween21and40DaysUsedForA2P2Nonfinancial : Total Value of Issues, with a Maturity Between 21 and 40 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Millions of Dollars) (NONFIN2140A2P2AMT)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween1and4DaysUsedForAANonfinancial : Number of Issues, with a Maturity Between 1 and 4 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Number) (NONFIN14AAVOL)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween10And20DaysUsedForA2P2Nonfinancial : Number of Issues, with a Maturity Between 10 and 20 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Number) (NONFIN1020A2P2VOL)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween10And20DaysUsedForAANonfinancial : Total Value of Issues, with a Maturity Between 10 and 20 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Millions of Dollars) (NONFIN1020AAAMT)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween21and40DaysUsedForAAAssetBacked : Total Value of Issues, with a Maturity Between 21 and 40 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Millions of Dollars) (AB2140AAAMT)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween10And20DaysUsedForAANonfinancial : Number of Issues, with a Maturity Between 10 and 20 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Number) (NONFIN1020AAVOL)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween1and4DaysUsedForA2P2Nonfinancial : Total Value of Issues, with a Maturity Between 1 and 4 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Millions of Dollars) (NONFIN14A2P2AMT)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween1and4DaysUsedForAANonfinancial : Total Value of Issues, with a Maturity Between 1 and 4 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Millions of Dollars) (NONFIN14AAAMT)
- Fred.CommercialPaper.TotalValueofCommercialPaperIssueswithaMaturityBetween1and4Days : Total Value of

Commercial Paper Issues with a Maturity Between 1 and 4 Days (in Millions of Dollars) (MKT14MKTAMT)

- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween10And20DaysUsedForA2P2Nonfinancial : Total Value of Issues, with a Maturity Between 10 and 20 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Millions of Dollars) (NONFIN1020A2P2AMT)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityGreaterThan80DaysUsedForAAFinancial : Number of Issues, with a Maturity Greater Than 80 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Number) (FINGT80AAVOL)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween10And20DaysUsedForAAFinancial : Number of Issues, with a Maturity Between 10 and 20 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Number) (FIN1020AAVOL)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween1and4DaysUsedForAAFinancial : Total Value of Issues, with a Maturity Between 1 and 4 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Millions of Dollars) (FIN14AAAMT)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween1and4DaysUsedForAAFinancial : Number of Issues, with a Maturity Between 1 and 4 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Number) (FIN14AAVOL)
- Fred.CommercialPaper.TotalValueofCommercialPaperIssueswithaMaturityBetween10And20Days : Total Value of Commercial Paper Issues with a Maturity Between 10 and 20 Days (in Millions of Dollars) (MKT1020MKTAMT)
- Fred.CommercialPaper.NumberofCommercialPaperIssueswithaMaturityBetween10And20Days : Number of Commercial Paper Issues with a Maturity Between 10 and 20 Days (in Number) (MKT1020MKTVOL)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween21and40DaysUsedForAAFinancial : Total Value of Issues, with a Maturity Between 21 and 40 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Millions of Dollars) (FIN2140AAAMT)
- Fred.CommercialPaper.NumberofCommercialPaperIssueswithaMaturityBetween1and4Days : Number of Commercial Paper Issues with a Maturity Between 1 and 4 Days (in Number) (MKT14MKTVOL)
- Fred.CommercialPaper.TotalValueofIssuersofCommercialPaperwithaMaturityBetween21and40Days : Total Value of Issuers of Commercial Paper with a Maturity Between 21 and 40 Days (in Millions of Dollars) (MKT2140MKTAMT)
- Fred.CommercialPaper.NumberofCommercialPaperIssueswithaMaturityBetween21and40Days : Number of Commercial Paper Issues with a Maturity Between 21 and 40 Days (in Number) (MKT2140MKTVOL)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween21and40DaysUsedForAAFinancial : Number of Issues, with a Maturity Between 21 and 40 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Number) (FIN2140AAVOL)
- Fred.CommercialPaper.TotalValueofIssuersofCommercialPaperwithaMaturityBetween41and80Days : Total Value of Issuers of Commercial Paper with a Maturity Between 41 and 80 Days (in Millions of Dollars) (MKT4180MKTAMT)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween5and9DaysUsedForAANonfinancial : Total Value of Issues, with a Maturity Between 5 and 9 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Millions of Dollars) (NONFIN59AAAMT)
- Fred.CommercialPaper.NumberofCommercialPaperIssueswithaMaturityBetween41and80Days : Number of Commercial Paper Issues with a Maturity Between 41 and 80 Days (in Number) (MKT4180MKTVOL)
- Fred.CommercialPaper.NumberofCommercialPaperIssueswithaMaturityBetween5and9Days : Number of Commercial Paper Issues with a Maturity Between 5 and 9 Days (in Number) (MKT59MKTVOL)
- Fred.CommercialPaper.TotalValueofIssuersofCommercialPaperwithaMaturityGreaterThan80Days : Total Value of Issuers of Commercial Paper with a Maturity Greater Than 80 Days (in Millions of Dollars) (MKTGT80MKTAMT)
- Fred.CommercialPaper.NumberofCommercialPaperIssueswithaMaturityGreaterThan80Days : Number of Commercial Paper Issues with a Maturity Greater Than 80 Days (in Number) (MKTGT80MKTVOL)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween41and80DaysUsedForAAFinancial : Total Value of Issues, with a Maturity Between 41 and 80 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Millions

of Dollars) (FIN4180AAAMT)

- Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween41and80DaysUsedForAAFinancial : Number of Issues, with a Maturity Between 41 and 80 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Number) (FIN4180AAVOL)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween41and80DaysUsedForAAAssetBacked : Total Value of Issues, with a Maturity Between 41 and 80 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Millions of Dollars) (AB4180AAAMT)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween5and9DaysUsedForAAFinancial : Total Value of Issues, with a Maturity Between 5 and 9 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Millions of Dollars) (FIN59AAAMT)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween5and9DaysUsedForAAFinancial : Number of Issues, with a Maturity Between 5 and 9 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Number) (FIN59AAVOL)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityGreaterThan80DaysUsedForAAFinancial : Total Value of Issues, with a Maturity Greater Than 80 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Millions of Dollars) (FINGT80AAAMT)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween10And20DaysUsedForAAFinancial : Total Value of Issues, with a Maturity Between 10 and 20 Days, Used in Calculating the AA Financial Commercial Paper Rates (in Millions of Dollars) (FIN1020AAAMT)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween21and40DaysUsedForAAAssetBacked : Number of Issues, with a Maturity Between 21 and 40 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Number) (AB2140AAVOL)
- Fred.CommercialPaper.TotalValueofIssuersofCommercialPaperwithaMaturityBetween5and9Days : Total Value of Issuers of Commercial Paper with a Maturity Between 5 and 9 Days (in Millions of Dollars) (MKT59MKTAMT)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityGreaterThan80DaysUsedForAAAssetBacked : Number of Issues, with a Maturity Greater Than 80 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Number) (ABGT80AAVOL)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween5and9DaysUsedForAANonfinancial : Number of Issues, with a Maturity Between 5 and 9 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Number) (NONFIN59AAVOL)
- Fred.CommercialPaper.FifteenDayAAAssetbackedCommercialPaperInterestRate : 15-Day AA Asset-backed Commercial Paper Interest Rate (in Percent) (RIFSPPAAAD15NB)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween5and9DaysUsedForAAAssetBacked : Total Value of Issues, with a Maturity Between 5 and 9 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Millions of Dollars) (AB59AAAMT)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween41and80DaysUsedForAAAssetBacked : Number of Issues, with a Maturity Between 41 and 80 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Number) (AB4180AAVOL)
- Fred.CommercialPaper.FifteenDayA2P2NonfinancialCommercialPaperInterestRate : 15-Day A2/P2 Nonfinancial Commercial Paper Interest Rate (in Percent) (RIFSPPNA2P2D15NB)
- Fred.CommercialPaper.SevenDayA2P2NonfinancialCommercialPaperInterestRate : 7-Day A2/P2 Nonfinancial Commercial Paper Interest Rate (in Percent) (RIFSPPNA2P2D07NB)
- Fred.CommercialPaper.OvernightA2P2NonfinancialCommercialPaperInterestRate : Overnight A2/P2 Nonfinancial Commercial Paper Interest Rate (in Percent) (RIFSPPNA2P2D01NB)
- Fred.CommercialPaper.NinetyDayAAFinancialCommercialPaperInterestRate : 90-Day AA Financial Commercial Paper Interest Rate (in Percent) (RIFSPPFAAD90NB)

- Fred.CommercialPaper.OvernightAAAssetbackedCommercialPaperInterestRate : Overnight AA Asset-backed Commercial Paper Interest Rate (in Percent) (RIFSPPAAAD01NB)
- Fred.CommercialPaper.Three0DayA2P2NonfinancialCommercialPaperInterestRate : 30-Day A2/P2 Nonfinancial Commercial Paper Interest Rate (in Percent) (RIFSPPNA2P2D30NB)
- Fred.CommercialPaper.SixtyDayAAFinancialCommercialPaperInterestRate : 60-Day AA Financial Commercial Paper Interest Rate (in Percent) (RIFSPPFAAD60NB)
- Fred.CommercialPaper.Three0DayAAFinancialCommercialPaperInterestRate : 30-Day AA Financial Commercial Paper Interest Rate (in Percent) (RIFSPPFAAD30NB)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityGreaterThan80DaysUsedForA2P2Nonfinancial : Total Value of Issues, with a Maturity Greater Than 80 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Millions of Dollars) (NONFINGT80A2P2AMT)
- Fred.CommercialPaper.Three0DayAAAssetbackedCommercialPaperInterestRate : 30-Day AA Asset-backed Commercial Paper Interest Rate (in Percent) (RIFSPPAAAD30NB)
- Fred.CommercialPaper.SixtyDayAAAssetbackedCommercialPaperInterestRate : 60-Day AA Asset-backed Commercial Paper Interest Rate (in Percent) (RIFSPPAAAD60NB)
- Fred.CommercialPaper.NinetyDayAAAssetbackedCommercialPaperInterestRate : 90-Day AA Asset-backed Commercial Paper Interest Rate (in Percent) (RIFSPPAAAD90NB)
- Fred.CommercialPaper.FifteenDayAAFinancialCommercialPaperInterestRate : 15-Day AA Financial Commercial Paper Interest Rate (in Percent) (RIFSPPFAAD15NB)
- Fred.CommercialPaper.SevenDayAAFinancialCommercialPaperInterestRate : 7-Day AA Financial Commercial Paper Interest Rate (in Percent) (RIFSPPFAAD07NB)
- Fred.CommercialPaper.SevenDayAAAssetbackedCommercialPaperInterestRate : 7-Day AA Asset-backed Commercial Paper Interest Rate (in Percent) (RIFSPPAAAD07NB)
- Fred.CommercialPaper.OvernightAAFinancialCommercialPaperInterestRate : Overnight AA Financial Commercial Paper Interest Rate (in Percent) (RIFSPPFAAD01NB)
- Fred.CommercialPaper.SixtyDayA2P2NonfinancialCommercialPaperInterestRate : 60-Day A2/P2 Nonfinancial Commercial Paper Interest Rate (in Percent) (RIFSPPNA2P2D60NB)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween5and9DaysUsedForAAAssetBacked : Number of Issues, with a Maturity Between 5 and 9 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Number) (AB59AAVOL)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween1and4DaysUsedForAAAssetBacked : Number of Issues, with a Maturity Between 1 and 4 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Number) (AB14AAVOL)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityGreaterThan80DaysUsedForA2P2Nonfinancial : Number of Issues, with a Maturity Greater Than 80 Days, Used in Calculating the A2/P2 Nonfinancial Commercial Paper Rates (in Number) (NONFINGT80A2P2VOL)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween1and4DaysUsedForAAAssetBacked : Total Value of Issues, with a Maturity Between 1 and 4 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Millions of Dollars) (AB14AAAMT)
- Fred.CommercialPaper.NinetyDayA2P2NonfinancialCommercialPaperInterestRate : 90-Day A2/P2 Nonfinancial Commercial Paper Interest Rate (in Percent) (RIFSPPNA2P2D90NB)
- Fred.CommercialPaper.NumberOfIssuesWithMaturityBetween10And20DaysUsedForAAAssetBacked : Number of Issues, with a Maturity Between 10 and 20 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Number) (AB1020AAVOL)
- Fred.CommercialPaper.TotalValueOfIssuesWithMaturityGreaterThan80DaysUsedForAANonfinancial : Total Value of

Issues, with a Maturity Greater Than 80 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Millions of Dollars) (`NONFINGT80AAAMT`)

- `Fred.CommercialPaper.OvernightAANonfinancialCommercialPaperInterestRate` : Overnight AA Nonfinancial Commercial Paper Interest Rate (in Percent) (`RIFSPPNAAD01NB`)
- `Fred.CommercialPaper.TotalValueOfIssuesWithMaturityBetween10And20DaysUsedForAAAssetBacked` : Total Value of Issues, with a Maturity Between 10 and 20 Days, Used in Calculating the AA Asset-Backed Commercial Paper Rates (in Millions of Dollars) (`AB1020AAAMT`)
- `Fred.CommercialPaper.SevenDayAANonfinancialCommercialPaperInterestRate` : 7-Day AA Nonfinancial Commercial Paper Interest Rate (in Percent) (`RIFSPPNAAD07NB`)
- `Fred.CommercialPaper.NinetyDayAANonfinancialCommercialPaperInterestRate` : 90-Day AA Nonfinancial Commercial Paper Interest Rate (in Percent) (`RIFSPPNAAD90NB`)
- `Fred.CommercialPaper.FifteenDayAANonfinancialCommercialPaperInterestRate` : 15-Day AA Nonfinancial Commercial Paper Interest Rate (in Percent) (`RIFSPPNAAD15NB`)
- `Fred.CommercialPaper.Three0DayAANonfinancialCommercialPaperInterestRate` : 30-Day AA Nonfinancial Commercial Paper Interest Rate (in Percent) (`RIFSPPNAAD30NB`)
- `Fred.CommercialPaper.SixtyDayAANonfinancialCommercialPaperInterestRate` : 60-Day AA Nonfinancial Commercial Paper Interest Rate (in Percent) (`RIFSPPNAAD60NB`)
- `Fred.CommercialPaper.NumberOfIssuesWithMaturityGreaterThan80DaysUsedForAANonfinancial` : Number of Issues, with a Maturity Greater Than 80 Days, Used in Calculating the AA Nonfinancial Commercial Paper Rates (in Number) (`NONFINGT80AAVOL`)
- `Fred.CommercialPaper.ThreeMonthCommercialPaperMinusFederalFundsRate` : 3-Month Commercial Paper Minus Federal Funds Rate (in Percent) (`CPFF`)

ICEB of AML

- `Fred.ICEBofAML.AAAAEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML AAA-A Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEM1BRRAAA2ACRPITRIV`)
- `Fred.ICEBofAML.AAAAUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML AAA-A US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEM1RAAA2ALCRPIUSTRIV`)
- `Fred.ICEBofAML.AsiaEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML Asia Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEMRACRPIASIATRIV`)
- `Fred.ICEBofAML.AsiaUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML Asia US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEMALLCRPIASIAUSTRIV`)
- `Fred.ICEBofAML.BandLowerEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML B and Lower Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEM4BRRBLCRPITRIV`)
- `Fred.ICEBofAML.BandLowerUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML B and Lower US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEM4RBLLCRPIUSTRIV`)
- `Fred.ICEBofAML.BBEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML BB Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEM3BRRBBCRPITRIV`)
- `Fred.ICEBofAML.BBUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML BB US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEM3RBBLCRPIUSTRIV`)
- `Fred.ICEBofAML.BBBEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML BBB Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEM2BRRBBBCRPITRIV`)
- `Fred.ICEBofAML.BBBUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML BBB US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEM2RBBBLCRPIUSTRIV`)

- `Fred.ICEBofAML.CrossoverEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML Crossover Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEM5BCOCRPITRIV`)
- `Fred.ICEBofAML.CrossoverUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML Crossover US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEMXOCOLCRPIUSTRIV`)
- `Fred.ICEBofAML.EmergingMarketsCorporatePlusIndexTotalReturnIndexValue` : ICE BofAML Emerging Markets Corporate Plus Index Total Return Index Value (in Index) (`BAMLEMCBPITRIV`)
- `Fred.ICEBofAML.EuroEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML Euro Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEMEBCRPIETRIV`)
- `Fred.ICEBofAML.EMEAEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML Europe, the Middle East, and Africa (EMEA) Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEMRECRPIEMEATRIV`)
- `Fred.ICEBofAML.EMEAUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML Europe, the Middle East, and Africa (EMEA) US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEMELLCRPIEMEAUSTRIV`)
- `Fred.ICEBofAML.FinancialEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML Financial Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEMFSFCRPITRIV`)
- `Fred.ICEBofAML.FinancialUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML Financial US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEMFLFLCRPIUSTRIV`)
- `Fred.ICEBofAML.HighGradeEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML High Grade Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEMIBHGCRPITRIV`)
- `Fred.ICEBofAML.HighGradeUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML High Grade US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEMHGHGLCRPIUSTRIV`)
- `Fred.ICEBofAML.HighYieldEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML High Yield Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEMHBHYCRPITRIV`)
- `Fred.ICEBofAML.HighYieldUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML High Yield US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEMHYHYLCRPIUSTRIV`)
- `Fred.ICEBofAML.LatinAmericaEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML Latin America Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEMRLCRPILATRIV`)
- `Fred.ICEBofAML.LatinAmericaUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML Latin America US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEMLLLCRPILAUSTRIV`)
- `Fred.ICEBofAML.NonFinancialEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML Non-Financial Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEMNSNFCRPITRIV`)
- `Fred.ICEBofAML.NonFinancialUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue` : ICE BofAML Non-Financial US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index) (`BAMLEMNFNFLCRPIUSTRIV`)
- `Fred.ICEBofAML.USCorporateMasterOptionAdjustedSpread` : ICE BofAML US Corporate Master Option-Adjusted Spread (in Percent) (`BAMLC0A0CM`)
- `Fred.ICEBofAML.USHighYieldMasterIIOptionAdjustedSpread` : ICE BofAML US High Yield Master II Option-Adjusted Spread (in Percent) (`BAMLH0A0HYM2`)
- `Fred.ICEBofAML.USCorporate1To3YearOptionAdjustedSpread` : ICE BofAML US Corporate 1-3 Year Option-Adjusted

Spread (in Percent) (BAMLC1A0C13Y)

- Fred.ICEBofAML.USCorporate10To15YearOptionAdjustedSpread : ICE BofAML US Corporate 10-15 Year Option-Adjusted Spread (in Percent) (BAMLC7A0C1015Y)
- Fred.ICEBofAML.USCorporateMoreThan15YearOptionAdjustedSpread : ICE BofAML US Corporate 15+ Year Option-Adjusted Spread (in Percent) (BAMLC8A0C15PY)
- Fred.ICEBofAML.USCorporate3To5YearOptionAdjustedSpread : ICE BofAML US Corporate 3-5 Year Option-Adjusted Spread (in Percent) (BAMLC2A0C35Y)
- Fred.ICEBofAML.USCorporate5To7YearOptionAdjustedSpread : ICE BofAML US Corporate 5-7 Year Option-Adjusted Spread (in Percent) (BAMLC3A0C57Y)
- Fred.ICEBofAML.USCorporate7To10YearOptionAdjustedSpread : ICE BofAML US Corporate 7-10 Year Option-Adjusted Spread (in Percent) (BAMLC4A0C710Y)
- Fred.ICEBofAML.PublicSectorIssuersUSEmergingMarketsLiquidCorporatePlusSubIndexTotalReturnIndexValue : ICE BofAML Public Sector Issuers US Emerging Markets Liquid Corporate Plus Sub-Index Total Return Index Value (in Index) (BAMLEMPUPUBSLCRPIUSTRIV)
- Fred.ICEBofAML.USEmergingMarketsCorporatePlusSubIndexTotalReturnIndexValue : ICE BofAML US Emerging Markets Corporate Plus Sub-Index Total Return Index Value (in Index) (BAMLEMUBCRPIUSTRIV)
- Fred.ICEBofAML.USEmergingMarketsLiquidCorporatePlusIndexTotalReturnIndexValue : ICE BofAML US Emerging Markets Liquid Corporate Plus Index Total Return Index Value (in Index) (BAMLEMCLLCRPIUSTRIV)
- Fred.ICEBofAML.EuroHighYieldIndexTotalReturnIndexValue : ICE BofAML Euro High Yield Index Total Return Index Value (in Index) (BAMLHE00EHYITRIV)
- Fred.ICEBofAML.USCorp1To3YearsTotalReturnIndexValue : ICE BofAML US Corp 1-3yr Total Return Index Value (in Index) (BAMLCC1A013YTRIV)
- Fred.ICEBofAML.USCorp10To15TotalReturnIndexValue : ICE BofAML US Corp 10-15yr Total Return Index Value (in Index) (BAMLCC7A01015YTRIV)
- Fred.ICEBofAML.USCorpMoreThan15YearsTotalReturnIndexValue : ICE BofAML US Corp 15+yr Total Return Index Value (in Index) (BAMLCC8A015PYTRIV)
- Fred.ICEBofAML.USCorpeTo5YearsTotalReturnIndexValue : ICE BofAML US Corp 3-5yr Total Return Index Value (in Index) (BAMLCC2A035YTRIV)
- Fred.ICEBofAML.USCorp5To7YearsTotalReturnIndexValue : ICE BofAML US Corp 5-7yr Total Return Index Value (in Index) (BAMLCC3A057YTRIV)
- Fred.ICEBofAML.USCorporate7To10YearsTotalReturnIndexValue : ICE BofAML US Corporate 7-10yr Total Return Index Value (in Index) (BAMLCC4A0710YTRIV)
- Fred.ICEBofAML.USCorpATotalReturnIndexValue : ICE BofAML US Corp A Total Return Index Value (in Index) (BAMLCC0A3ATRIV)
- Fred.ICEBofAML.USCorpAATotalReturnIndexValue : ICE BofAML US Corp AA Total Return Index Value (in Index) (BAMLCC0A2AATRIV)
- Fred.ICEBofAML.USCorpAAATotalReturnIndexValue : ICE BofAML US Corp AAA Total Return Index Value (in Index) (BAMLCC0A1AAATRIV)
- Fred.ICEBofAML.USHighYieldBTotalReturnIndexValue : ICE BofAML US High Yield B Total Return Index Value (in Index) (BAMLHYH0A2BTRIV)
- Fred.ICEBofAML.USHighYieldBBTotalReturnIndexValue : ICE BofAML US High Yield BB Total Return Index Value (in Index) (BAMLHYH0A1BBTRIV)
- Fred.ICEBofAML.USCorpBBBTotalReturnIndexValue : ICE BofAML US Corp BBB Total Return Index Value (in Index) (BAMLCC0A4BBBTRIV)
- Fred.ICEBofAML.USHighYieldCCCorBelowTotalReturnIndexValue : ICE BofAML US High Yield CCC or Below Total Return

Index Value (in Index) (BAMLHYH0A3CMTRIV)

- Fred.ICEBofAML.USCorpMasterTotalReturnIndexValue : ICE BofAML US Corp Master Total Return Index Value (in Index) (BAMLCC0A0CMTRIV)
- Fred.ICEBofAML.USHighYieldMasterIITotalReturnIndexValue : ICE BofAML US High Yield Master II Total Return Index Value (in Index) (BAMLHYH0A0HYM2TRIV)
- Fred.ICEBofAML.AAAAEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML AAA-A Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEM1BRRAAA2ACRPIOAS)
- Fred.ICEBofAML.AAAAUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML AAA-A US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEM1RAAA2ALCRPIUSOAS)
- Fred.ICEBofAML.AsiaEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML Asia Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEMRACRPIASIAOAS)
- Fred.ICEBofAML.AsiaUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML Asia US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEMALLCRPIASIAUSOAS)
- Fred.ICEBofAML.BandLowerEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML B and Lower Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEM4BRRBLCRPIOAS)
- Fred.ICEBofAML.BandLowerUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML B and Lower US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEM4RBLLCRPIUSOAS)
- Fred.ICEBofAML.BBEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML BB Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEM3BRRBBCRPIOAS)
- Fred.ICEBofAML.BBUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML BB US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEM3RBBLCRPIUSOAS)
- Fred.ICEBofAML.BBBEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML BBB Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEM2BRRBBBCRPIOAS)
- Fred.ICEBofAML.BBBUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML BBB US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEM2RBBBLCRPIUSOAS)
- Fred.ICEBofAML.CrossoverEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML Crossover Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEM5BCOCRPIOAS)
- Fred.ICEBofAML.CrossoverUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML Crossover US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEMXOCOLCRPIUSOAS)
- Fred.ICEBofAML.EmergingMarketsCorporatePlusIndexOptionAdjustedSpread : ICE BofAML Emerging Markets Corporate Plus Index Option-Adjusted Spread (in Percent) (BAMLEMCBPIOAS)
- Fred.ICEBofAML.EuroEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML Euro Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEMEBCRPIEOAS)
- Fred.ICEBofAML.EMEAEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML Europe, the Middle East, and Africa (EMEA) Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEMRECRPIEMEAOAS)
- Fred.ICEBofAML.EMEAUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML Europe, the Middle East, and Africa (EMEA) US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEMELLCRPIEMEAUSOAS)
- Fred.ICEBofAML.FinancialEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML Financial Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEMFSFCRPIOAS)
- Fred.ICEBofAML.FinancialUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML Financial US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (

BAMLEMFLFLCRPIUSOAS)

- Fred.ICEBofAML.HighGradeEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML High Grade Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEMIBHGCRPIOAS)
- Fred.ICEBofAML.HighGradeUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML High Grade US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEMHGHGLCRPIUSOAS)
- Fred.ICEBofAML.HighYieldEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML High Yield Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEMHBHYCRPIOAS)
- Fred.ICEBofAML.HighYieldUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML High Yield US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEMHYHYLCRPIUSOAS)
- Fred.ICEBofAML.LatinAmericaEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML Latin America Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEMRLCRPILAOAS)
- Fred.ICEBofAML.LatinAmericaUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML Latin America US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEMLLLCRPILAUSOAS)
- Fred.ICEBofAML.NonFinancialEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML Non-Financial Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEMNSNFCRPIOAS)
- Fred.ICEBofAML.NonFinancialUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML Non-Financial US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEMNFNFLCRPIUSOAS)
- Fred.ICEBofAML.PublicSectorIssuersUSEmergingMarketsLiquidCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML Public Sector Issuers US Emerging Markets Liquid Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEMPUPUBSLCRPIUSOAS)
- Fred.ICEBofAML.USEmergingMarketsCorporatePlusSubIndexOptionAdjustedSpread : ICE BofAML US Emerging Markets Corporate Plus Sub-Index Option-Adjusted Spread (in Percent) (BAMLEMUBCRPIUSOAS)
- Fred.ICEBofAML.USEmergingMarketsLiquidCorporatePlusIndexOptionAdjustedSpread : ICE BofAML US Emerging Markets Liquid Corporate Plus Index Option-Adjusted Spread (in Percent) (BAMLEMCLLCRPIUSOAS)
- Fred.ICEBofAML.EuroHighYieldIndexOptionAdjustedSpread : ICE BofAML Euro High Yield Index Option-Adjusted Spread (in Percent) (BAMLHE00EHYIOAS)
- Fred.ICEBofAML.USCorporateAOptionAdjustedSpread : ICE BofAML US Corporate A Option-Adjusted Spread (in Percent) (BAMLC0A3CA)
- Fred.ICEBofAML.USCorporateAAOptionAdjustedSpread : ICE BofAML US Corporate AA Option-Adjusted Spread (in Percent) (BAMLC0A2CAA)
- Fred.ICEBofAML.USCorporateAAAOptionAdjustedSpread : ICE BofAML US Corporate AAA Option-Adjusted Spread (in Percent) (BAMLC0A1CAAA)
- Fred.ICEBofAML.USHighYieldBOptionAdjustedSpread : ICE BofAML US High Yield B Option-Adjusted Spread (in Percent) (BAMLH0A2HYB)
- Fred.ICEBofAML.USHighYieldBBOptionAdjustedSpread : ICE BofAML US High Yield BB Option-Adjusted Spread (in Percent) (BAMLH0A1HYBB)
- Fred.ICEBofAML.USCorporateBBBOptionAdjustedSpread : ICE BofAML US Corporate BBB Option-Adjusted Spread (in Percent) (BAMLC0A4CBBB)
- Fred.ICEBofAML.USHighYieldCCCorBelowOptionAdjustedSpread : ICE BofAML US High Yield CCC or Below Option-Adjusted Spread (in Percent) (BAMLH0A3HYC)
- Fred.ICEBofAML.AAAAEmergingMarketsCorporatePlusSubIndexEffectiveYield : ICE BofAML AAA-A Emerging Markets

Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEM1BRRAAA2ACRPIEY)

- Fred.ICEBofAML.AAAAUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield : ICE BofAML AAA-A US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEM1RAAA2ALCRPIUSEY)
- Fred.ICEBofAML.AsiaEmergingMarketsCorporatePlusSubIndexEffectiveYield : ICE BofAML Asia Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEMRACRPIASIAEY)
- Fred.ICEBofAML.AsiaUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield : ICE BofAML Asia US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEMALLCRPIASIAUSEY)
- Fred.ICEBofAML.BandLowerEmergingMarketsCorporatePlusSubIndexEffectiveYield : ICE BofAML B and Lower Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEM4BRRBLCRPIEY)
- Fred.ICEBofAML.BandLowerUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield : ICE BofAML B and Lower US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEM4RBLLCRPIUSEY)
- Fred.ICEBofAML.BBEmergingMarketsCorporatePlusSubIndexEffectiveYield : ICE BofAML BB Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEM3BRRBBCRPIEY)
- Fred.ICEBofAML.BBUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield : ICE BofAML BB US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEM3RBBLCRPIUSEY)
- Fred.ICEBofAML.BBBEmergingMarketsCorporatePlusSubIndexEffectiveYield : ICE BofAML BBB Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEM2BRRBBBCRPIEY)
- Fred.ICEBofAML.BBBUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield : ICE BofAML BBB US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEM2RBBBLCRPIUSEY)
- Fred.ICEBofAML.CrossoverEmergingMarketsCorporatePlusSubIndexEffectiveYield : ICE BofAML Crossover Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEM5BCOCRPIEY)
- Fred.ICEBofAML.CrossoverUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield : ICE BofAML Crossover US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEMXOCOLCRPIUSEY)
- Fred.ICEBofAML.EmergingMarketsCorporatePlusIndexEffectiveYield : ICE BofAML Emerging Markets Corporate Plus Index Effective Yield (in Percent) (BAMLEMCBPIEY)
- Fred.ICEBofAML.EuroEmergingMarketsCorporatePlusSubIndexEffectiveYield : ICE BofAML Euro Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEMEBCRPIEEY)
- Fred.ICEBofAML.EuroHighYieldIndexEffectiveYield : ICE BofAML Euro High Yield Index Effective Yield (in Percent) (BAMLHE00EHYIEY)
- Fred.ICEBofAML.EMEAEmergingMarketsCorporatePlusSubIndexEffectiveYield : ICE BofAML Europe, the Middle East, and Africa (EMEA) Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEMRECRPIEMEAEY)
- Fred.ICEBofAML.EMEAUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield : ICE BofAML Europe, the Middle East, and Africa (EMEA) US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEMELLCRPIEMEAUSEY)
- Fred.ICEBofAML.FinancialEmergingMarketsCorporatePlusSubIndexEffectiveYield : ICE BofAML Financial Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEMFSFCRPIEY)
- Fred.ICEBofAML.FinancialUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield : ICE BofAML Financial US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEMFLFLCRPIUSEY)
- Fred.ICEBofAML.HighGradeEmergingMarketsCorporatePlusSubIndexEffectiveYield : ICE BofAML High Grade Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEMIBHGCRPIEY)
- Fred.ICEBofAML.HighGradeUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield : ICE BofAML High Grade US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEMHGHGLCRPIUSEY)
- Fred.ICEBofAML.HighYieldEmergingMarketsCorporatePlusSubIndexEffectiveYield : ICE BofAML High Yield Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEMHBHYCRPIEY)
- Fred.ICEBofAML.HighYieldUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield : ICE BofAML High Yield

US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEMHYHYLCRPIUSEY)

- Fred.ICEBofAML.LatinAmericaEmergingMarketsCorporatePlusSubIndexEffectiveYield : ICE BofAML Latin America Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEMRLCRPILAEY)
- Fred.ICEBofAML.LatinAmericaUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield : ICE BofAML Latin America US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEMLLLCRPILAUSEY)
- Fred.ICEBofAML.NonFinancialEmergingMarketsCorporatePlusSubIndexEffectiveYield : ICE BofAML Non-Financial Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEMNSNFCRPIEY)
- Fred.ICEBofAML.NonFinancialUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield : ICE BofAML Non-Financial US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEMNFNFLCRPIUSEY)
- Fred.ICEBofAML.PublicSectorIssuersUSEmergingMarketsLiquidCorporatePlusSubIndexEffectiveYield : ICE BofAML Public Sector Issuers US Emerging Markets Liquid Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEMPUPUBSLCRPIUSEY)
- Fred.ICEBofAML.USCorporate1ThreeYearEffectiveYield : ICE BofAML US Corporate 1-3 Year Effective Yield (in Percent) (BAMLC1A0C13YEY)
- Fred.ICEBofAML.USCorporate10To15YearEffectiveYield : ICE BofAML US Corporate 10-15 Year Effective Yield (in Percent) (BAMLC7A0C1015YEY)
- Fred.ICEBofAML.USCorporateMoreThan15YearEffectiveYield : ICE BofAML US Corporate 15+ Year Effective Yield (in Percent) (BAMLC8A0C15PYEY)
- Fred.ICEBofAML.USCorporate3To5YearEffectiveYield : ICE BofAML US Corporate 3-5 Year Effective Yield (in Percent) (BAMLC2A0C35YEY)
- Fred.ICEBofAML.USCorporate5To7YearEffectiveYield : ICE BofAML US Corporate 5-7 Year Effective Yield (in Percent) (BAMLC3A0C57YEY)
- Fred.ICEBofAML.USCorporate7To10YearEffectiveYield : ICE BofAML US Corporate 7-10 Year Effective Yield (in Percent) (BAMLC4A0C710YEY)
- Fred.ICEBofAML.USCorporateAEffectiveYield : ICE BofAML US Corporate A Effective Yield (in Percent) (BAMLC0A3CAEY)
- Fred.ICEBofAML.USCorporateAAEffectiveYield : ICE BofAML US Corporate AA Effective Yield (in Percent) (BAMLC0A2CAAEY)
- Fred.ICEBofAML.USCorporateAAAEffectiveYield : ICE BofAML US Corporate AAA Effective Yield (in Percent) (BAMLC0A1CAAAEY)
- Fred.ICEBofAML.USHighYieldBEffectiveYield : ICE BofAML US High Yield B Effective Yield (in Percent) (BAMLH0A2HYBEY)
- Fred.ICEBofAML.USHighYieldBBEffectiveYield : ICE BofAML US High Yield BB Effective Yield (in Percent) (BAMLH0A1HYBBEY)
- Fred.ICEBofAML.USCorporateBBBEffectiveYield : ICE BofAML US Corporate BBB Effective Yield (in Percent) (BAMLC0A4CBBBEY)
- Fred.ICEBofAML.USHighYieldCCCorBelowEffectiveYield : ICE BofAML US High Yield CCC or Below Effective Yield (in Percent) (BAMLH0A3HYCEY)
- Fred.ICEBofAML.USCorporateMasterEffectiveYield : ICE BofAML US Corporate Master Effective Yield (in Percent) (BAMLC0A0CMEY)
- Fred.ICEBofAML.USEmergingMarketsCorporatePlusSubIndexEffectiveYield : ICE BofAML US Emerging Markets Corporate Plus Sub-Index Effective Yield (in Percent) (BAMLEMUBCRPIUSEY)
- Fred.ICEBofAML.USEmergingMarketsLiquidCorporatePlusIndexEffectiveYield : ICE BofAML US Emerging Markets Liquid Corporate Plus Index Effective Yield (in Percent) (BAMLEMCLLCRPIUSEY)
- Fred.ICEBofAML.USHighYieldMasterIIEffectiveYield : ICE BofAML US High Yield Master II Effective Yield (in Percent) (BAMLH0A0HYM2EY)

- Fred.ICEBofAML.AAAAEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML AAA-A Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEM1BRRAAA2ACRPISYTW)
- Fred.ICEBofAML.AAAAUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML AAA-A US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEM1RAAA2ALCRPIUSSYTW)
- Fred.ICEBofAML.AsiaEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML Asia Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMRACRPIASIASYTW)
- Fred.ICEBofAML.AsiaUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML Asia US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMALLCRPIASIAUSSYTW)
- Fred.ICEBofAML.BandLowerEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML B and Lower Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEM4BRRBLCRPISYTW)
- Fred.ICEBofAML.BandLowerUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML B and Lower US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEM4RBLLCRPIUSSYTW)
- Fred.ICEBofAML.BBEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML BB Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEM3BRRBBCRPISYTW)
- Fred.ICEBofAML.BBUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML BB US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEM3RBBLCRPIUSSYTW)
- Fred.ICEBofAML.BBBEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML BBB Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEM2BRRBBBCRPISYTW)
- Fred.ICEBofAML.BBBUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML BBB US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEM2RBBBLCRPIUSSYTW)
- Fred.ICEBofAML.CrossoverEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML Crossover Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEM5BCOCRPISYTW)
- Fred.ICEBofAML.CrossoverUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML Crossover US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMXOCOLCRPIUSSYTW)
- Fred.ICEBofAML.EmergingMarketsCorporatePlusIndexSemiAnnualYieldtoWorst : ICE BofAML Emerging Markets Corporate Plus Index Semi-Annual Yield to Worst (in Percent) (BAMLEMCBPISYTW)
- Fred.ICEBofAML.EuroEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML Euro Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMEBCRPIESYTW)
- Fred.ICEBofAML.EuroHighYieldIndexSemiAnnualYieldtoWorst : ICE BofAML Euro High Yield Index Semi-Annual Yield to Worst (in Percent) (BAMLHE00EHYISYTW)
- Fred.ICEBofAML.EMEAEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML Europe, the Middle East, and Africa (EMEA) Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMRECRPIEMEASYTW)
- Fred.ICEBofAML.EMEAUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML Europe, the Middle East, and Africa (EMEA) US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMELLCRPIEMEAUSSYTW)
- Fred.ICEBofAML.FinancialEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML Financial Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMFSFCRPISYTW)
- Fred.ICEBofAML.FinancialUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML Financial US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMFLFLCRPIUSSYTW)
- Fred.ICEBofAML.HighGradeEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML High Grade

Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMIBHGCRPISYTW)

- Fred.ICEBofAML.HighGradeUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML High Grade US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMHGHGLCRPIUSSYTW)
- Fred.ICEBofAML.HighYieldEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML High Yield Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMHBHYCRPISYTW)
- Fred.ICEBofAML.HighYieldUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML High Yield US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMHYHYLCRPIUSSYTW)
- Fred.ICEBofAML.LatinAmericaEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML Latin America Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMRLCRPILASYTW)
- Fred.ICEBofAML.LatinAmericaUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML Latin America US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMLLLCRPILAUSSYTW)
- Fred.ICEBofAML.NonFinancialEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML Non-Financial Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMNSNFCRPISYTW)
- Fred.ICEBofAML.NonFinancialUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML Non-Financial US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMNFNFLCRPIUSSYTW)
- Fred.ICEBofAML.PrivateSectorIssuersEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML Private Sector Issuers Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMPTPRVICRPISYTW)
- Fred.ICEBofAML.PrivateSectorIssuersUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML Private Sector Issuers US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMPVPRIVSLCRPIUSSYTW)
- Fred.ICEBofAML.PublicSectorIssuersEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML Public Sector Issuers Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMPBPUBSICRPISYTW)
- Fred.ICEBofAML.PublicSectorIssuersUSEmergingMarketsLiquidCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML Public Sector Issuers US Emerging Markets Liquid Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMPUPUBSLCRPIUSSYTW)
- Fred.ICEBofAML.USCorporate1To3YearSemiAnnualYieldtoWorst : ICE BofAML US Corporate 1-3 Year Semi-Annual Yield to Worst (in Percent) (BAMLC1A0C13YSYTW)
- Fred.ICEBofAML.USCorporate10To15YearSemiAnnualYieldtoWorst : ICE BofAML US Corporate 10-15 Year Semi-Annual Yield to Worst (in Percent) (BAMLC7A0C1015YSYTW)
- Fred.ICEBofAML.USCorporateMoreThan15YearSemiAnnualYieldtoWorst : ICE BofAML US Corporate 15+ Year Semi-Annual Yield to Worst (in Percent) (BAMLC8A0C15PYSYTW)
- Fred.ICEBofAML.USCorporate3To5YearSemiAnnualYieldtoWorst : ICE BofAML US Corporate 3-5 Year Semi-Annual Yield to Worst (in Percent) (BAMLC2A0C35YSYTW)
- Fred.ICEBofAML.USCorporate5To7YearSemiAnnualYieldtoWorst : ICE BofAML US Corporate 5-7 Year Semi-Annual Yield to Worst (in Percent) (BAMLC3A0C57YSYTW)
- Fred.ICEBofAML.USCorporate7To10YearSemiAnnualYieldtoWorst : ICE BofAML US Corporate 7-10 Year Semi-Annual Yield to Worst (in Percent) (BAMLC4A0C710YSYTW)
- Fred.ICEBofAML.USCorporateASemiAnnualYieldtoWorst : ICE BofAML US Corporate A Semi-Annual Yield to Worst (in Percent) (BAMLC0A3CASYTW)

- Fred.ICEBofAML.USCorporateAASemiAnnualYieldtoWorst : ICE BofAML US Corporate AA Semi-Annual Yield to Worst (in Percent) (BAMLC0A2CAASYTW)
- Fred.ICEBofAML.USCorporateAAASemiAnnualYieldtoWorst : ICE BofAML US Corporate AAA Semi-Annual Yield to Worst (in Percent) (BAMLC0A1CAAASYTW)
- Fred.ICEBofAML.USHighYieldBSemiAnnualYieldtoWorst : ICE BofAML US High Yield B Semi-Annual Yield to Worst (in Percent) (BAMLH0A2HYBSYTW)
- Fred.ICEBofAML.USHighYieldBBSemiAnnualYieldtoWorst : ICE BofAML US High Yield BB Semi-Annual Yield to Worst (in Percent) (BAMLH0A1HYBBSYTW)
- Fred.ICEBofAML.USCorporateBBBSemiAnnualYieldtoWorst : ICE BofAML US Corporate BBB Semi-Annual Yield to Worst (in Percent) (BAMLC0A4CBBBSYTW)
- Fred.ICEBofAML.USHighYieldCCCorBelowSemiAnnualYieldtoWorst : ICE BofAML US High Yield CCC or Below Semi-Annual Yield to Worst (in Percent) (BAMLH0A3HYCSYTW)
- Fred.ICEBofAML.USCorporateMasterSemiAnnualYieldtoWorst : ICE BofAML US Corporate Master Semi-Annual Yield to Worst (in Percent) (BAMLC0A0CMSYTW)
- Fred.ICEBofAML.USEmergingMarketsCorporatePlusSubIndexSemiAnnualYieldtoWorst : ICE BofAML US Emerging Markets Corporate Plus Sub-Index Semi-Annual Yield to Worst (in Percent) (BAMLEMUBCRPIUSSYTW)
- Fred.ICEBofAML.USEmergingMarketsLiquidCorporatePlusIndexSemiAnnualYieldtoWorst : ICE BofAML US Emerging Markets Liquid Corporate Plus Index Semi-Annual Yield to Worst (in Percent) (BAMLEMCLLCRPIUSSYTW)
- Fred.ICEBofAML.USHighYieldMasterIISemiAnnualYieldtoWorst : ICE BofAML US High Yield Master II Semi-Annual Yield to Worst (in Percent) (BAMLH0A0HYM2SYTW)

LIBOR

- Fred.LIBOR.SpotNextBasedOnSwissFranc : Spot Next London Interbank Offered Rate (LIBOR), based on Swiss Franc (in Percent) (CHFONTD156N)
- Fred.LIBOR.SpotNextBasedOnJapaneseYen : Spot Next London Interbank Offered Rate (LIBOR), based on Japanese Yen (in Percent) (JPYONTD156N)
- Fred.LIBOR.SixMonthBasedOnJapaneseYen : 6-Month London Interbank Offered Rate (LIBOR), based on Japanese Yen (in Percent) (JPY6MTD156N)
- Fred.LIBOR.ThreeMonthBasedOnJapaneseYen : 3-Month London Interbank Offered Rate (LIBOR), based on Japanese Yen (in Percent) (JPY3MTD156N)
- Fred.LIBOR.SixMonthBasedOnUSD : 6-Month London Interbank Offered Rate (LIBOR), based on U.S. Dollar (in Percent) (USD6MTD156N)
- Fred.LIBOR.OneMonthBasedOnJapaneseYen : 1-Month London Interbank Offered Rate (LIBOR), based on Japanese Yen (in Percent) (JPY1MTD156N)
- Fred.LIBOR.TwelveMonthBasedOnJapaneseYen : 12-Month London Interbank Offered Rate (LIBOR), based on Japanese Yen (in Percent) (JPY12MD156N)
- Fred.LIBOR.TwelveMonthBasedOnBritishPound : 12-Month London Interbank Offered Rate (LIBOR), based on British Pound (in Percent) (GBP12MD156N)
- Fred.LIBOR.OneMonthBasedOnBritishPound : 1-Month London Interbank Offered Rate (LIBOR), based on British Pound (in Percent) (GBP1MTD156N)
- Fred.LIBOR.OneWeekBasedOnBritishPound : 1-Week London Interbank Offered Rate (LIBOR), based on British Pound (in Percent) (GBP1WKD156N)
- Fred.LIBOR.TwoMonthBasedOnBritishPound : 2-Month London Interbank Offered Rate (LIBOR), based on British Pound (in Percent) (GBP2MTD156N)
- Fred.LIBOR.ThreeMonthBasedOnBritishPound : 3-Month London Interbank Offered Rate (LIBOR), based on British Pound

(in Percent) (GBP3MTD156N)

- Fred.LIBOR.OneWeekBasedOnJapaneseYen : 1-Week London Interbank Offered Rate (LIBOR), based on Japanese Yen (in Percent) (JPY1WKD156N)
- Fred.LIBOR.TwoMonthBasedOnJapaneseYen : 2-Month London Interbank Offered Rate (LIBOR), based on Japanese Yen (in Percent) (JPY2MTD156N)
- Fred.LIBOR.SixMonthBasedOnSwissFranc : 6-Month London Interbank Offered Rate (LIBOR), based on Swiss Franc (in Percent) (CHF6MTD156N)
- Fred.LIBOR.ThreeMonthBasedOnSwissFranc : 3-Month London Interbank Offered Rate (LIBOR), based on Swiss Franc (in Percent) (CHF3MTD156N)
- Fred.LIBOR.OneMonthBasedOnUSD : 1-Month London Interbank Offered Rate (LIBOR), based on U.S. Dollar (in Percent) (USD1MTD156N)
- Fred.LIBOR.TwelveMonthBasedOnSwissFranc : 12-Month London Interbank Offered Rate (LIBOR), based on Swiss Franc (in Percent) (CHF12MD156N)
- Fred.LIBOR.TwelveMonthBasedOnUSD : 12-Month London Interbank Offered Rate (LIBOR), based on U.S. Dollar (in Percent) (USD12MD156N)
- Fred.LIBOR.OneMonthBasedOnSwissFranc : 1-Month London Interbank Offered Rate (LIBOR), based on Swiss Franc (in Percent) (CHF1MTD156N)
- Fred.LIBOR.OneWeekBasedOnSwissFranc : 1-Week London Interbank Offered Rate (LIBOR), based on Swiss Franc (in Percent) (CHF1WKD156N)
- Fred.LIBOR.TwoMonthBasedOnSwissFranc : 2-Month London Interbank Offered Rate (LIBOR), based on Swiss Franc (in Percent) (CHF2MTD156N)
- Fred.LIBOR.TwelveMonthBasedOnEuro : 12-Month London Interbank Offered Rate (LIBOR), based on Euro (in Percent) (EUR12MD156N)
- Fred.LIBOR.SixMonthBasedOnBritishPound : 6-Month London Interbank Offered Rate (LIBOR), based on British Pound (in Percent) (GBP6MTD156N)
- Fred.LIBOR.OneMonthBasedOnEuro : 1-Month London Interbank Offered Rate (LIBOR), based on Euro (in Percent) (EUR1MTD156N)
- Fred.LIBOR.TwoMonthBasedOnEuro : 2-Month London Interbank Offered Rate (LIBOR), based on Euro (in Percent) (EUR2MTD156N)
- Fred.LIBOR.ThreeMonthBasedOnEuro : 3-Month London Interbank Offered Rate (LIBOR), based on Euro (in Percent) (EUR3MTD156N)
- Fred.LIBOR.SixMonthBasedOnEuro : 6-Month London Interbank Offered Rate (LIBOR), based on Euro (in Percent) (EUR6MTD156N)
- Fred.LIBOR.OvernightBasedOnEuro : Overnight London Interbank Offered Rate (LIBOR), based on Euro (in Percent) (EURONTD156N)
- Fred.LIBOR.OneWeekBasedOnUSD : 1-Week London Interbank Offered Rate (LIBOR), based on U.S. Dollar (in Percent) (USD1WKD156N)
- Fred.LIBOR.TwoMonthBasedOnUSD : 2-Month London Interbank Offered Rate (LIBOR), based on U.S. Dollar (in Percent) (USD2MTD156N)
- Fred.LIBOR.ThreeMonthBasedOnUSD : 3-Month London Interbank Offered Rate (LIBOR), based on U.S. Dollar (in Percent) (USD3MTD156N)
- Fred.LIBOR.OvernightBasedOnUSD : Overnight London Interbank Offered Rate (LIBOR), based on U.S. Dollar (in Percent) (USDONTD156N)
- Fred.LIBOR.OneWeekBasedOnEuro : 1-Week London Interbank Offered Rate (LIBOR), based on Euro (in Percent) (EUR1WKD156N)

- `Fred.LIBOR.OvernightBasedOnBritishPound` : Overnight London Interbank Offered Rate (LIBOR), based on British Pound (in Percent) (`GBPONTD156N`)

OECD Recession Indicators

- `Fred.OECDRecessionIndicators.FourBigEuropeanCountriesFromPeakThroughTheTrough` : OECD based Recession Indicators for Four Big European Countries from the Peak through the Trough (in +1 or 0) (`4BIGEURORECDM`)
- `Fred.OECDRecessionIndicators.AustraliaFromPeakThroughTheTrough` : OECD based Recession Indicators for Australia from the Peak through the Trough (in +1 or 0) (`AUSRECDM`)
- `Fred.OECDRecessionIndicators.AustriaFromPeakThroughTheTrough` : OECD based Recession Indicators for Austria from the Peak through the Trough (in +1 or 0) (`AUTRECDM`)
- `Fred.OECDRecessionIndicators.BelgiumFromPeakThroughTheTrough` : OECD based Recession Indicators for Belgium from the Peak through the Trough (in +1 or 0) (`BELRECDM`)
- `Fred.OECDRecessionIndicators.BrazilFromPeakThroughTheTrough` : OECD based Recession Indicators for Brazil from the Peak through the Trough (in +1 or 0) (`BRARECDM`)
- `Fred.OECDRecessionIndicators.CanadaFromPeakThroughTheTrough` : OECD based Recession Indicators for Canada from the Peak through the Trough (in +1 or 0) (`CANRECDM`)
- `Fred.OECDRecessionIndicators.SwitzerlandFromPeakThroughTheTrough` : OECD based Recession Indicators for Switzerland from the Peak through the Trough (in +1 or 0) (`CHERECDM`)
- `Fred.OECDRecessionIndicators.ChileFromPeakThroughTheTrough` : OECD based Recession Indicators for Chile from the Peak through the Trough (in +1 or 0) (`CHLRECDM`)
- `Fred.OECDRecessionIndicators.ChinaFromPeakThroughTheTrough` : OECD based Recession Indicators for China from the Peak through the Trough (in +1 or 0) (`CHNRECDM`)
- `Fred.OECDRecessionIndicators.CzechRepublicFromPeakThroughTheTrough` : OECD based Recession Indicators for the Czech Republic from the Peak through the Trough (in +1 or 0) (`CZERECDM`)
- `Fred.OECDRecessionIndicators.GermanyFromPeakThroughTheTrough` : OECD based Recession Indicators for Germany from the Peak through the Trough (in +1 or 0) (`DEURECDM`)
- `Fred.OECDRecessionIndicators.DenmarkFromPeakThroughTheTrough` : OECD based Recession Indicators for Denmark from the Peak through the Trough (in +1 or 0) (`DNKRECDM`)
- `Fred.OECDRecessionIndicators.SpainFromPeakThroughTheTrough` : OECD based Recession Indicators for Spain from the Peak through the Trough (in +1 or 0) (`ESPRECDM`)
- `Fred.OECDRecessionIndicators.EstoniaFromPeakThroughTheTrough` : OECD based Recession Indicators for Estonia from the Peak through the Trough (in +1 or 0) (`ESTRECDM`)
- `Fred.OECDRecessionIndicators.EuroAreaFromPeakThroughTheTrough` : Retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/ESTRECDM (`EURORECDM`)
- `Fred.OECDRecessionIndicators.FinlandFromPeakThroughTheTrough` : OECD based Recession Indicators for Finland from the Peak through the Trough (in +1 or 0) (`FINRECDM`)
- `Fred.OECDRecessionIndicators.FranceFromPeakThroughTheTrough` : OECD based Recession Indicators for France from the Peak through the Trough (in +1 or 0) (`FRARECDM`)
- `Fred.OECDRecessionIndicators.UnitedKingdomFromPeakThroughTheTrough` : OECD based Recession Indicators for the United Kingdom from the Peak through the Trough (in +1 or 0) (`GBRRECDM`)
- `Fred.OECDRecessionIndicators.GreeceFromPeakThroughTheTrough` : OECD based Recession Indicators for Greece from the Peak through the Trough (in +1 or 0) (`GRCRECDM`)
- `Fred.OECDRecessionIndicators.HungaryFromPeakThroughTheTrough` : OECD based Recession Indicators for Hungary from the Peak through the Trough (in +1 or 0) (`HUNRECDM`)
- `Fred.OECDRecessionIndicators.IndonesiaFromPeakThroughTheTrough` : OECD based Recession Indicators for Indonesia

from the Peak through the Trough (in +1 or 0) (IDNRECDM)

- Fred.OECDRecessionIndicators.IndiaFromPeakThroughTheTrough : OECD based Recession Indicators for India from the Peak through the Trough (in +1 or 0) (INDRECDM)
- Fred.OECDRecessionIndicators.IrelandFromPeakThroughTheTrough : OECD based Recession Indicators for Ireland from the Peak through the Trough (in +1 or 0) (IRLRECDM)
- Fred.OECDRecessionIndicators.IsraelFromPeakThroughTheTrough : OECD based Recession Indicators for Israel from the Peak through the Trough (in +1 or 0) (ISRRECDM)
- Fred.OECDRecessionIndicators.ItalyFromPeakThroughTheTrough : OECD based Recession Indicators for Italy from the Peak through the Trough (in +1 or 0) (ITARECDM)
- Fred.OECDRecessionIndicators.JapanFromPeakThroughTheTrough : OECD based Recession Indicators for Japan from the Peak through the Trough (in +1 or 0) (JPNRECDM)
- Fred.OECDRecessionIndicators.KoreaFromPeakThroughTheTrough : OECD based Recession Indicators for Korea from the Peak through the Trough (in +1 or 0) (KORRECDM)
- Fred.OECDRecessionIndicators.LuxembourgFromPeakThroughTheTrough : OECD based Recession Indicators for Luxembourg from the Peak through the Trough (in +1 or 0) (LUXRECDM)
- Fred.OECDRecessionIndicators.MajorFiveAsiaFromPeakThroughTheTrough : OECD based Recession Indicators for Major 5 Asia from the Peak through the Trough (in +1 or 0) (MAJOR5ASIARECDM)
- Fred.OECDRecessionIndicators.MexicoFromPeakThroughTheTrough : OECD based Recession Indicators for Mexico from the Peak through the Trough (in +1 or 0) (MEXRECDM)
- Fred.OECDRecessionIndicators.MajorSevenCountriesFromPeakThroughTheTrough : OECD based Recession Indicators for Major Seven Countries from the Peak through the Trough (in +1 or 0) (MSCRECDM)
- Fred.OECDRecessionIndicators.NAFTAAreaFromPeakThroughTheTrough : OECD based Recession Indicators for NAFTA Area from the Peak through the Trough (in +1 or 0) (NAFTARECDM)
- Fred.OECDRecessionIndicators.NetherlandsFromPeakThroughTheTrough : OECD based Recession Indicators for Netherlands from the Peak through the Trough (in +1 or 0) (NDLRECDM)
- Fred.OECDRecessionIndicators.NorwayFromPeakThroughTheTrough : OECD based Recession Indicators for Norway from the Peak through the Trough (in +1 or 0) (NORRECDM)
- Fred.OECDRecessionIndicators.NewZealandFromPeakThroughTheTrough : OECD based Recession Indicators for New Zealand from the Peak through the Trough (in +1 or 0) (NZLRECDM)
- Fred.OECDRecessionIndicators.OECDEuropeFromPeakThroughTheTrough : OECD based Recession Indicators for OECD Europe from the Peak through the Trough (in +1 or 0) (OECDEUROPERECDM)
- Fred.OECDRecessionIndicators.OECDAndNonmemberEconomiesFromPeakThroughTheTrough : OECD based Recession Indicators for OECD and Non-member Economies from the Peak through the Trough (in +1 or 0) (OECDNMERECDM)
- Fred.OECDRecessionIndicators.OECDTotalAreaFromPeakThroughTheTrough : OECD based Recession Indicators for the OECD Total Area from the Peak through the Trough (in +1 or 0) (OECDRECDM)
- Fred.OECDRecessionIndicators.PolandFromPeakThroughTheTrough : OECD based Recession Indicators for Poland from the Peak through the Trough (in +1 or 0) (POLRECDM)
- Fred.OECDRecessionIndicators.PortugalFromPeakThroughTheTrough : OECD based Recession Indicators for Portugal from the Peak through the Trough (in +1 or 0) (PRTRECDM)
- Fred.OECDRecessionIndicators.RussianFederationFromPeakThroughTheTrough : OECD based Recession Indicators for Russian Federation from the Peak through the Trough (in +1 or 0) (RUSRECDM)
- Fred.OECDRecessionIndicators.SlovakRepublicFromPeakThroughTheTrough : OECD based Recession Indicators for the Slovak Republic from the Peak through the Trough (in +1 or 0) (SVKRECDM)
- Fred.OECDRecessionIndicators.SloveniaFromPeakThroughTheTrough : OECD based Recession Indicators for Slovenia from the Peak through the Trough (in +1 or 0) (SVNRECDM)

- `Fred.OECDRecessionIndicators.SwedenFromPeakThroughTheTrough` : OECD based Recession Indicators for Sweden from the Peak through the Trough (in +1 or 0) (`SWERECDM`)
- `Fred.OECDRecessionIndicators.TurkeyFromPeakThroughTheTrough` : OECD based Recession Indicators for Turkey from the Peak through the Trough (in +1 or 0) (`TURRECDM`)
- `Fred.OECDRecessionIndicators.UnitedStatesFromPeakThroughTheTrough` : OECD based Recession Indicators for the United States from the Peak through the Trough (in +1 or 0) (`USARECDM`)
- `Fred.OECDRecessionIndicators.SouthAfricaFromPeakThroughTheTrough` : OECD based Recession Indicators for South Africa from the Peak through the Trough (in +1 or 0) (`ZAFRECDM`)
- `Fred.OECDRecessionIndicators.FourBigEuropeanCountriesFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Four Big European Countries from the Period following the Peak through the Trough (in +1 or 0) (`4BIGEURORECD`)
- `Fred.OECDRecessionIndicators.AustraliaFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Australia from the Period following the Peak through the Trough (in +1 or 0) (`AUSRECD`)
- `Fred.OECDRecessionIndicators.AustriaFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Austria from the Period following the Peak through the Trough (in +1 or 0) (`AUTRECD`)
- `Fred.OECDRecessionIndicators.BelgiumFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Belgium from the Period following the Peak through the Trough (in +1 or 0) (`BELRECD`)
- `Fred.OECDRecessionIndicators.BrazilFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Brazil from the Period following the Peak through the Trough (in +1 or 0) (`BRARECD`)
- `Fred.OECDRecessionIndicators.CanadaFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Canada from the Period following the Peak through the Trough (in +1 or 0) (`CANRECD`)
- `Fred.OECDRecessionIndicators.SwitzerlandFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Switzerland from the Period following the Peak through the Trough (in +1 or 0) (`CHERECD`)
- `Fred.OECDRecessionIndicators.ChileFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Chile from the Period following the Peak through the Trough (in +1 or 0) (`CHLRECD`)
- `Fred.OECDRecessionIndicators.ChinaFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for China from the Period following the Peak through the Trough (in +1 or 0) (`CHNRECD`)
- `Fred.OECDRecessionIndicators.CzechRepublicFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for the Czech Republic from the Period following the Peak through the Trough (in +1 or 0) (`CZERECD`)
- `Fred.OECDRecessionIndicators.GermanyFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Germany from the Period following the Peak through the Trough (in +1 or 0) (`DEURECD`)
- `Fred.OECDRecessionIndicators.DenmarkFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Denmark from the Period following the Peak through the Trough (in +1 or 0) (`DNKRECD`)
- `Fred.OECDRecessionIndicators.SpainFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Spain from the Period following the Peak through the Trough (in +1 or 0) (`ESPRECD`)
- `Fred.OECDRecessionIndicators.EstoniaFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Estonia from the Period following the Peak through the Trough (in +1 or 0) (`ESTRECD`)
- `Fred.OECDRecessionIndicators.EuroAreaFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Euro Area from the Period following the Peak through the Trough (in +1 or 0) (`EURORECD`)
- `Fred.OECDRecessionIndicators.FinlandFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Finland from the Period following the Peak through the Trough (in +1 or 0) (`FINRECD`)
- `Fred.OECDRecessionIndicators.FranceFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for France from the Period following the Peak through the Trough (in +1 or 0) (`FRARECD`)
- `Fred.OECDRecessionIndicators.UnitedKingdomFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for the United Kingdom from the Period following the Peak through the Trough (in +1 or 0) (`GBRRECD`)

- `Fred.OECDRecessionIndicators.GreeceFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Greece from the Period following the Peak through the Trough (in +1 or 0) (`GRCRECD`)
- `Fred.OECDRecessionIndicators.HungaryFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Hungary from the Period following the Peak through the Trough (in +1 or 0) (`HUNRECD`)
- `Fred.OECDRecessionIndicators.IndonesiaFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Indonesia from the Period following the Peak through the Trough (in +1 or 0) (`IDNRECD`)
- `Fred.OECDRecessionIndicators.IndiaFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for India from the Period following the Peak through the Trough (in +1 or 0) (`INDRECD`)
- `Fred.OECDRecessionIndicators.IrelandFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Ireland from the Period following the Peak through the Trough (in +1 or 0) (`IRLRECD`)
- `Fred.OECDRecessionIndicators.IsraelFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Israel from the Period following the Peak through the Trough (in +1 or 0) (`ISRRECD`)
- `Fred.OECDRecessionIndicators.ItalyFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Italy from the Period following the Peak through the Trough (in +1 or 0) (`ITARECD`)
- `Fred.OECDRecessionIndicators.JapanFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Japan from the Period following the Peak through the Trough (in +1 or 0) (`JPNRECD`)
- `Fred.OECDRecessionIndicators.KoreaFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Korea from the Period following the Peak through the Trough (in +1 or 0) (`KORRECD`)
- `Fred.OECDRecessionIndicators.LuxembourgFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Luxembourg from the Period following the Peak through the Trough (in +1 or 0) (`LUXRECD`)
- `Fred.OECDRecessionIndicators.MajorFiveAsiaFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Major 5 Asia from the Period following the Peak through the Trough (in +1 or 0) (`MAJOR5ASIARECD`)
- `Fred.OECDRecessionIndicators.MexicoFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Mexico from the Period following the Peak through the Trough (in +1 or 0) (`MEXRECD`)
- `Fred.OECDRecessionIndicators.MajorSevenCountriesFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Major Seven Countries from the Period following the Peak through the Trough (in +1 or 0) (`MSCRECD`)
- `Fred.OECDRecessionIndicators.NAFTAAreaFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for NAFTA Area from the Period following the Peak through the Trough (in +1 or 0) (`NAFTARECD`)
- `Fred.OECDRecessionIndicators.NetherlandsFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Netherlands from the Period following the Peak through the Trough (in +1 or 0) (`NDLRECD`)
- `Fred.OECDRecessionIndicators.NorwayFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Norway from the Period following the Peak through the Trough (in +1 or 0) (`NORRECD`)
- `Fred.OECDRecessionIndicators.NewZealandFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for New Zealand from the Period following the Peak through the Trough (in +1 or 0) (`NZLRECD`)
- `Fred.OECDRecessionIndicators.OECDEuropeFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for OECD Europe from the Period following the Peak through the Trough (in +1 or 0) (`OECDEUROPERECD`)
- `Fred.OECDRecessionIndicators.OECDandNonmemberEconomiesFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for OECD and Non-member Economies from the Period following the Peak through the Trough (in +1 or 0) (`OECDNMERECD`)
- `Fred.OECDRecessionIndicators.OECDTotalAreaFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for the OECD Total Area from the Period following the Peak through the Trough (in +1 or 0) (`OECDRECD`)
- `Fred.OECDRecessionIndicators.PolandFromPeriodFollowingPeakThroughTheTrough` : OECD based Recession Indicators for Poland from the Period following the Peak through the Trough (in +1 or 0) (`POLRECD`)
- `Fred.OECDRecessionIndicators.PortugalFromPeriodFollowingPeakThroughTheTrough` : Retrieved from FRED, Federal

Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/POLRECD (PRTRECD)

- Fred.OECDRecessionIndicators.RussianFederationFromPeriodFollowingPeakThroughTheTrough : OECD based Recession Indicators for Russian Federation from the Period following the Peak through the Trough (in +1 or 0) (RUSRECD)
- Fred.OECDRecessionIndicators.SlovakRepublicFromPeriodFollowingPeakThroughTheTrough : OECD based Recession Indicators for the Slovak Republic from the Period following the Peak through the Trough (in +1 or 0) (SVKRECD)
- Fred.OECDRecessionIndicators.SloveniaFromPeriodFollowingPeakThroughTheTrough : OECD based Recession Indicators for Slovenia from the Period following the Peak through the Trough (in +1 or 0) (SVNRECD)
- Fred.OECDRecessionIndicators.SwedenFromPeriodFollowingPeakThroughTheTrough : OECD based Recession Indicators for Sweden from the Period following the Peak through the Trough (in +1 or 0) (SWERECD)
- Fred.OECDRecessionIndicators.TurkeyFromPeriodFollowingPeakThroughTheTrough : OECD based Recession Indicators for Turkey from the Period following the Peak through the Trough (in +1 or 0) (TURRECD)
- Fred.OECDRecessionIndicators.UnitedStatesFromPeriodFollowingPeakThroughTheTrough : OECD based Recession Indicators for the United States from the Period following the Peak through the Trough (in +1 or 0) (USARECD)
- Fred.OECDRecessionIndicators.SouthAfricaFromPeriodFollowingPeakThroughTheTrough : OECD based Recession Indicators for South Africa from the Period following the Peak through the Trough (in +1 or 0) (ZAFRECD)
- Fred.OECDRecessionIndicators.FourBigEuropeanCountriesFromPeakThroughThePeriodPrecedingtheTrough : OECD based Recession Indicators for Four Big European Countries from the Peak through the Period preceding the Trough (in +1 or 0) (4BIGEURORECDP)
- Fred.OECDRecessionIndicators.AustraliaFromPeakThroughThePeriodPrecedingtheTrough : OECD based Recession Indicators for Australia from the Peak through the Period preceding the Trough (in +1 or 0) (AUSRECDP)
- Fred.OECDRecessionIndicators.AustriaFromPeakThroughThePeriodPrecedingtheTrough : OECD based Recession Indicators for Austria from the Peak through the Period preceding the Trough (in +1 or 0) (AUTRECDP)
- Fred.OECDRecessionIndicators.BelgiumFromPeakThroughThePeriodPrecedingtheTrough : OECD based Recession Indicators for Belgium from the Peak through the Period preceding the Trough (in +1 or 0) (BELRECDP)
- Fred.OECDRecessionIndicators.BrazilFromPeakThroughThePeriodPrecedingtheTrough : OECD based Recession Indicators for Brazil from the Peak through the Period preceding the Trough (in +1 or 0) (BRARECDP)
- Fred.OECDRecessionIndicators.CanadaFromPeakThroughThePeriodPrecedingtheTrough : OECD based Recession Indicators for Canada from the Peak through the Period preceding the Trough (in +1 or 0) (CANRECDP)
- Fred.OECDRecessionIndicators.SwitzerlandFromPeakThroughThePeriodPrecedingtheTrough : OECD based Recession Indicators for Switzerland from the Peak through the Period preceding the Trough (in +1 or 0) (CHERECDP)
- Fred.OECDRecessionIndicators.ChileFromPeakThroughThePeriodPrecedingtheTrough : OECD based Recession Indicators for Chile from the Peak through the Period preceding the Trough (in +1 or 0) (CHLRECDP)
- Fred.OECDRecessionIndicators.ChinaFromPeakThroughThePeriodPrecedingtheTrough : OECD based Recession Indicators for China from the Peak through the Period preceding the Trough (in +1 or 0) (CHNRECDP)
- Fred.OECDRecessionIndicators.CzechRepublicFromPeakThroughThePeriodPrecedingtheTrough : OECD based Recession Indicators for the Czech Republic from the Peak through the Period preceding the Trough (in +1 or 0) (CZERECDP)
- Fred.OECDRecessionIndicators.GermanyFromPeakThroughThePeriodPrecedingtheTrough : OECD based Recession Indicators for Germany from the Peak through the Period preceding the Trough (in +1 or 0) (DEURECDP)
- Fred.OECDRecessionIndicators.DenmarkFromPeakThroughThePeriodPrecedingtheTrough : OECD based Recession Indicators for Denmark from the Peak through the Period preceding the Trough (in +1 or 0) (DNKRECDP)
- Fred.OECDRecessionIndicators.SpainFromPeakThroughThePeriodPrecedingtheTrough : OECD based Recession Indicators for Spain from the Peak through the Period preceding the Trough (in +1 or 0) (ESPRECDP)
- Fred.OECDRecessionIndicators.EstoniaFromPeakThroughThePeriodPrecedingtheTrough : OECD based Recession Indicators for Estonia from the Peak through the Period preceding the Trough (in +1 or 0) (ESTRECDP)

- `Fred.OECDRecessionIndicators.EuroAreaFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Euro Area from the Peak through the Period preceding the Trough (in +1 or 0) (`EURORECDP`)
- `Fred.OECDRecessionIndicators.FinlandFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Finland from the Peak through the Period preceding the Trough (in +1 or 0) (`FINRECDP`)
- `Fred.OECDRecessionIndicators.FranceFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for France from the Peak through the Period preceding the Trough (in +1 or 0) (`FRARECDP`)
- `Fred.OECDRecessionIndicators.UnitedKingdomFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for the United Kingdom from the Peak through the Period preceding the Trough (in +1 or 0) (`GBRRECDP`)
- `Fred.OECDRecessionIndicators.GreeceFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Greece from the Peak through the Period preceding the Trough (in +1 or 0) (`GRCRECDP`)
- `Fred.OECDRecessionIndicators.HungaryFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Hungary from the Peak through the Period preceding the Trough (in +1 or 0) (`HUNRECDP`)
- `Fred.OECDRecessionIndicators.IndonesiaFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Indonesia from the Peak through the Period preceding the Trough (in +1 or 0) (`IDNRECDP`)
- `Fred.OECDRecessionIndicators.IndiaFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for India from the Peak through the Period preceding the Trough (in +1 or 0) (`INDRECDP`)
- `Fred.OECDRecessionIndicators.IrelandFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Ireland from the Peak through the Period preceding the Trough (in +1 or 0) (`IRLRECDP`)
- `Fred.OECDRecessionIndicators.IsraelFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Israel from the Peak through the Period preceding the Trough (in +1 or 0) (`ISRRECDP`)
- `Fred.OECDRecessionIndicators.ItalyFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Italy from the Peak through the Period preceding the Trough (in +1 or 0) (`ITARECDP`)
- `Fred.OECDRecessionIndicators.JapanFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Japan from the Peak through the Period preceding the Trough (in +1 or 0) (`JPNRECDP`)
- `Fred.OECDRecessionIndicators.KoreaFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Korea from the Peak through the Period preceding the Trough (in +1 or 0) (`KORRECDP`)
- `Fred.OECDRecessionIndicators.LuxembourgFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Luxembourg from the Peak through the Period preceding the Trough (in +1 or 0) (`LUXRECDP`)
- `Fred.OECDRecessionIndicators.MajorFiveAsiaFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Major 5 Asia from the Peak through the Period preceding the Trough (in +1 or 0) (`MAJOR5ASIARECDP`)
- `Fred.OECDRecessionIndicators.MexicoFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Mexico from the Peak through the Period preceding the Trough (in +1 or 0) (`MEXRECDP`)
- `Fred.OECDRecessionIndicators.MajorSevenCountriesFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Major Seven Countries from the Peak through the Period preceding the Trough (in +1 or 0) (`MSCRECDP`)
- `Fred.OECDRecessionIndicators.NAFTAAreaFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for NAFTA Area from the Peak through the Period preceding the Trough (in +1 or 0) (`NAFTARECDP`)
- `Fred.OECDRecessionIndicators.NetherlandsFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Netherlands from the Peak through the Period preceding the Trough (in +1 or 0) (`NDLRECDP`)
- `Fred.OECDRecessionIndicators.NorwayFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Norway from the Peak through the Period preceding the Trough (in +1 or 0) (`NORRECDP`)
- `Fred.OECDRecessionIndicators.NewZealandFromPeakThroughThePeriodPrecedingtheTrough` : Retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/NORRECDP (`NZLRECDP`)

- `Fred.OECDRecessionIndicators.OECDEuropeFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for OECD Europe from the Peak through the Period preceding the Trough (in +1 or 0) (`OECDEUROPERECDP`)
- `Fred.OECDRecessionIndicators.OECDandNonmemberEconomiesFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for OECD and Non-member Economies from the Peak through the Period preceding the Trough (in +1 or 0) (`OECDNMERECDP`)
- `Fred.OECDRecessionIndicators.OECDTotalAreaFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for the OECD Total Area from the Peak through the Period preceding the Trough (in +1 or 0) (`OECDRECDP`)
- `Fred.OECDRecessionIndicators.PolandFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Poland from the Peak through the Period preceding the Trough (in +1 or 0) (`POLRECDP`)
- `Fred.OECDRecessionIndicators.PortugalFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Portugal from the Peak through the Period preceding the Trough (in +1 or 0) (`PRTRECDP`)
- `Fred.OECDRecessionIndicators.RussianFederationFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Russian Federation from the Peak through the Period preceding the Trough (in +1 or 0) (`RUSRECDP`)
- `Fred.OECDRecessionIndicators.SlovakRepublicFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for the Slovak Republic from the Peak through the Period preceding the Trough (in +1 or 0) (`SVKRECDP`)
- `Fred.OECDRecessionIndicators.SloveniaFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Slovenia from the Peak through the Period preceding the Trough (in +1 or 0) (`SVNRECDP`)
- `Fred.OECDRecessionIndicators.SwedenFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Sweden from the Peak through the Period preceding the Trough (in +1 or 0) (`SWERECDP`)
- `Fred.OECDRecessionIndicators.TurkeyFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for Turkey from the Peak through the Period preceding the Trough (in +1 or 0) (`TURRECDP`)
- `Fred.OECDRecessionIndicators.UnitedStatesFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for the United States from the Peak through the Period preceding the Trough (in +1 or 0) (`USARECDP`)
- `Fred.OECDRecessionIndicators.SouthAfricaFromPeakThroughThePeriodPrecedingtheTrough` : OECD based Recession Indicators for South Africa from the Peak through the Period preceding the Trough (in +1 or 0) (`ZAFRECDP`)

Trade Weighted Indexes

- `Fred.TradeWeightedIndexes.MajorCurrenciesGoods` : Trade Weighted U.S. Dollar Index: Major Currencies, Goods (in Index Mar 1973=100) (`DTWEXM`)
- `Fred.TradeWeightedIndexes.OtherImportantTradingPartnersGoods` : Trade Weighted U.S. Dollar Index: Other Important Trading Partners, Goods (in Index Jan 1997=100) (`DTWEXO`)
- `Fred.TradeWeightedIndexes.BroadGoods` : Trade Weighted U.S. Dollar Index: Broad, Goods (in Index Jan 1997=100) (`DTWEXB`)
- `Fred.TradeWeightedIndexes.AdvancedForeignEconomiesGoodsAndServices` : Trade Weighted U.S. Dollar Index: Advanced Foreign Economies, Goods and Services (in Index Jan 2006=100) (`DTWEXAFEGS`)
- `Fred.TradeWeightedIndexes.BroadGoodsAndServices` : Trade Weighted U.S. Dollar Index: Broad, Goods and Services (in Index Jan 2006=100) (`DTWEXBGS`)
- `Fred.TradeWeightedIndexes.EmergingMarketsEconomiesGoodsAndServices` : Trade Weighted U.S. Dollar Index: Emerging Markets Economies, Goods and Services (in Index Jan 2006=100) (`DTWEXEMEGS`)

Example Applications

The FRED dataset enables you to accurately design strategies utilizing macroeconomic indicators. Examples include the

following strategies:

- Trading on macroeconomic factors
- Macroeconomic risk modeling

Classic Algorithm Example

The following example algorithm buys SPY when the FRED Peak to Trough value is 1. Otherwise, it holds cash.

```C#
public class FredAlternativeDataAlgorithm : QCAlgorithm
{
    private Symbol _fredPeakToTrough;
    private Symbol _spy;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        _spy = AddEquity("SPY", Resolution.Daily).Symbol;

        // Requesting FED US peak-to-trough OECD recession indicators for trade signal generation
        _fredPeakToTrough = AddData<Fred>(Fred.OECDRecessionIndicators.UnitedStatesFromPeakThroughTheTrough).Symbol;

        // Historical data
        var history = History<Fred>(_fredPeakToTrough, 60, Resolution.Daily);
        Debug($"We got {history.Count()} items from our history request");
    }

    public override void OnData(Slice slice)
    {
        // Trade with updated FED peak-to-trough indicator
        if (slice.ContainsKey(_fredPeakToTrough) && slice.ContainsKey(_spy))
        {
            var peakToTrough = slice.Get<Fred>(_fredPeakToTrough).Value;

            // Buy SPY if peak to trough value is 0, which is the expansionary period
            if (peakToTrough == 0m && !Portfolio.Invested)
            {
                SetHoldings(_spy, 1);
            }

            // Liquidate holdings if peak to trough value is 1, which is recessionary period
            else if (peakToTrough == 1m && Portfolio.Invested)
            {
                Liquidate(_spy);
            }
        }
    }
}
```

Framework Algorithm Example

The following example algorithm buys SPY when the FRED Peak to Trough value is 1. Otherwise, it holds cash.

```C#
public class FredAlternativeDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Universe contains only SPY as the market representative
        AddUniverseSelection(
            new ManualUniverseSelectionModel(
                QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA)
        ));
        // Custom alpha model that use FED data as trade signals
        AddAlpha(new FredAlphaModel(this));
        // Invest equally to evenly diversify the capital concentration risk
```

```
        // Invest equally to evenly dissipate the capital concentration risk
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }

    public class FredAlphaModel : AlphaModel
    {
        private List<Symbol> _symbols = new List<Symbol>();
        private Symbol _fredPeakToTrough;
        private decimal? _fredPeakToTroughValue = null;

        public FredAlphaModel(QCAlgorithm algorithm)
        {
            // Requesting FED US peak-to-trough OECD recession indicators for trade signal generation
            _fredPeakToTrough = algorithm.AddData<Fred>
(Fred.OECDRecessionIndicators.UnitedStatesFromPeakThroughTheTrough, Resolution.Daily).Symbol;

            // Historical data
            var history = algorithm.History<Fred>(_fredPeakToTrough, 60, Resolution.Daily);
            algorithm.Debug($"We got {history.Count()} items from our history request");
            if (history.Count() == 0)
            {
                return;
            }
            _fredPeakToTroughValue = history.Last().Value;
        }

        public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
        {
            var insights = new List<Insight>();

            // Trade with updated FED peak-to-trough indicator
            if (slice.ContainsKey(_fredPeakToTrough))
            {
                _fredPeakToTroughValue = slice.Get<Fred>(_fredPeakToTrough).Value;
            }

            // Ensure we have a FRED peak to trough value
            if (_fredPeakToTroughValue == null)
            {
                return insights;
            }

            foreach (var symbol in _symbols)
            {
                // Ensure we have data for the security in the current Slice to avoid stale filling
                if (!(slice.ContainsKey(symbol) && slice[symbol] != null))
                {
                    continue;
                }

                // Buy SPY if peak to trough value is 0, which is the expansionary period
                if (_fredPeakToTroughValue == 0m && !algorithm.Portfolio[symbol].Invested)
                {
                    insights.Add(Insight.Price(symbol, TimeSpan.FromDays(365*2),  InsightDirection.Up));
                }
                // Liquidate holdings if peak to trough value is 1, which is recessionary period
                else if (_fredPeakToTroughValue == 1m && algorithm.Portfolio[symbol].Invested)
                {
                    insights.Add(Insight.Price(symbol, TimeSpan.FromDays(365*2),  InsightDirection.Flat));
                }
            }

            return insights;
        }

        public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
        {
            foreach (var security in changes.AddedSecurities)
            {
                _symbols.Add(security.Symbol);
            }

            foreach (var security in changes.RemovedSecurities)
            {
                _symbols.Remove(security.Symbol);
            }
        }
    }
}
```

Data Point Attributes

The FRED dataset provides `Fred` objects, which have the following attributes:

Datasets

Federal Reserve Bank of St Louis

The Federal Reserve Bank of St. Louis, often referred to as the St. Louis Fed, is one of the 12 regional banks that make up the United States Federal Reserve System. It is responsible for the Eighth Federal Reserve District, which includes the states of Arkansas, Illinois, Indiana, Kentucky, Mississippi, Missouri, and Tennessee, as well as portions of eastern Kansas and southern Illinois.

The St. Louis Fed, like other regional banks, participates in the formulation and implementation of monetary policy in the United States. It contributes to the Federal Open Market Committee (FOMC) meetings, where key decisions regarding interest rates and other monetary policy tools are made.

US Interest Rate

Federal Reserve Bank of St Louis

US Interest Rate

Introduction

The US Interest Rate dataset provides the primary credit rate from the Federal Open Market Committee (FOMC). The data starts in January 2003 and is updated on a daily frequency. This dataset is created using information from the FOMC meetings.

For more information about the US Interest Rate dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

The Federal Reserve Bank of St. Louis, often referred to as the St. Louis Fed, is one of the 12 regional banks that make up the United States Federal Reserve System. It is responsible for the Eighth Federal Reserve District, which includes the states of Arkansas, Illinois, Indiana, Kentucky, Mississippi, Missouri, and Tennessee, as well as portions of eastern Kansas and southern Illinois.

The St. Louis Fed, like other regional banks, participates in the formulation and implementation of monetary policy in the United States. It contributes to the Federal Open Market Committee (FOMC) meetings, where key decisions regarding interest rates and other monetary policy tools are made.

Getting Started

The following snippet demonstrates how to access data from the US Interest Rate dataset:

```C#
var interestRate = RiskFreeInterestRateModel.GetInterestRate(Time);
```

To find the average interest rate between two dates, call the `GetRiskFreeRate` method.

```C#
var avgRiskFreeRate = RiskFreeInterestRateModel.GetRiskFreeRate(Time.AddDays(-365), Time);
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2003
Data Density	Sparse
Resolution	Daily
Timezone	New York

Requesting Data

You don't need any special code to request US Interest Rate data. `QCAlgorithm` automatically subscribes to the data by setting

its default risk free interest rate model .

Accessing Data

To get the current US Interest Rate data, call the `GetInterestRate` method of the `RiskFreeInterestRateModel` object with the current time.

```
C#
var interestRate = RiskFreeInterestRateModel.GetInterestRate(Time);
```

Historical Data

To get the average risk free interest rate for a window of time, call the `GetRiskFreeRate` method with the start date and end date.

```
C#
var riskFreeRate = RiskFreeInterestRateModel.GetRiskFreeRate(Time.AddDays(-365), Time);
```

To get the average risk free interest rate for a set of dates, call the `GetAverageRiskFreeRate` method with the list of dates.

```
C#
var riskFreeRate = RiskFreeInterestRateModel.GetAverageRiskFreeRate(
    new [] {Time, Time.AddDays(-180), Time.AddDays(-365)}
);
```

Example Applications

The US Interest Rate dataset provides an important economic indicator. Examples include the following applications:

- Accurately calculating indicators that are a function of the risk-free rate, like Sharpe ratios.
- Forming a portfolio of assets that have a history of outperforming when the rate is increasing/decreasing.
- Canceling orders in a Risk Management model when the expected return of the asset is less than the risk-free rate.

Classic Algorithm Example

The following example algorithm plots the current interest rate and the last year's average.

```
C#
public class RiskFreeInterestRateModelAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        AddEquity("SPY");
    }

    public override void OnEndOfDay(Symbol symbol)
    {
        SetHoldings(symbol, 1);
        // Plot the current interest rate and the 1-year average rate for comparison
        Plot("Interest Rate", "EOD", RiskFreeInterestRateModel.GetInterestRate(Time));
        // Get the average risk free rate of the last year at the current time
        Plot("Interest Rate", "1-year Window", RiskFreeInterestRateModel.GetRiskFreeRate(Time.AddDays(-365), Time));
    }
}
```

Datasets

Kavout

Kavout was created by ex-Googlers and the founding team used to work at Google, Microsoft, Baidu, and financial firms with a proven track record of building many mission-critical machine learning systems where billions of data points were processed in real-time to predict the best outcome for core search ranking, ads monetization, recommendations, and trading platforms.

Their mission is to build machine investing solutions to find alpha with adaptive learning algorithms and to create an edge by assimilating vast quantities of complex data through the latest AI and Machine Learning methods to generate signals to uncover hidden, dynamic, and nonlinear patterns in the financial markets.

Composite Factor Bundle

Kavout

Composite Factor Bundle

Introduction

The Composite Factor Bundle dataset by Kavout provides ensemble scores for popular market factors. Kavout signals are machine-learning enhanced scores that capture the returns of systematic factors such as quality, value, momentum, growth, and low volatility. There are many different anomalies discovered by researchers and practitioners across these factor categories and there is no good common definition of each style across the literature. Kavout creates an ensemble score for each style that gauges the different factors considered in the literature and industry practice.

In this data set, you will find Kavout's proprietary signals for quality, value, momentum, growth, and low volatility, which have been adopted by some of the multi-billion dollar quant funds in New York and London. Each signal is generated by an ensemble model consisting of inputs from hundreds of anomalies. The data is generated on a daily basis and covers all the stocks traded in US major markets such as NYSE and Nasdaq since 2003. You could leverage this abundant set of signals to construct and backtest your strategies.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Composite Factor Bundle dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Kavout was created by ex-Googlers and the founding team used to work at Google, Microsoft, Baidu, and financial firms with a proven track record of building many mission-critical machine learning systems where billions of data points were processed in real-time to predict the best outcome for core search ranking, ads monetization, recommendations, and trading platforms.

Their mission is to build machine investing solutions to find alpha with adaptive learning algorithms and to create an edge by assimilating vast quantities of complex data through the latest AI and Machine Learning methods to generate signals to uncover hidden, dynamic, and nonlinear patterns in the financial markets.

Getting Started

The following snippet demonstrates how to request data from the Composite Factor Bundle dataset:

```C#
_symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
_datasetSymbol = AddData<KavoutCompositeFactorBundle>(_symbol).Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2003
Asset Coverage	8,000 US Equities
Data Density	Regular
Resolution	Daily
Timezone	UTC

Requesting Data

To add Composite Factor Bundle data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```
public class KavoutCompositeFactorBundleAlgorithm : QCAlgorithm { private Symbol _symbol, _datasetSym
bol; public override void Initialize() { SetStartDate(2019, 1, 1); SetEndDate(2020, 6, 1); SetCash(10
0000); _symbol = AddEquity("AAPL", Resolution.Daily).Symbol; _datasetSymbol = AddData & lt; KavoutCom
positeFactorBundle & gt; (_symbol).Symbol; } }
```

Accessing Data

To get the current Composite Factor Bundle data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

C#

```
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var dataPoint = slice[_datasetSymbol];
        Log($"{_datasetSymbol} momentum at {slice.Time}: {dataPoint.Momentum}");
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

C#

```
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<KavoutCompositeFactorBundle>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} momentum at {slice.Time}: {dataPoint.Momentum}");
    }
}
```

Historical Data

To get historical Composite Factor Bundle data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
var history = History<KavoutCompositeFactorBundle>(_datasetSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```
RemoveSecurity(_datasetSymbol);
```

If you subscribe to Composite Factor Bundle data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Composite Factor Bundle dataset enables you to access the performance of 5 different factors in order to engineer strategies. Examples include the following strategies:

- Performing return-risk optimization based on performance and volatility scoring.
- Weighing stocks based on regression analysis in factor-vector space.

Classic Algorithm Example

The following example algorithm creates a dynamic universe of the 100 most liquid US Equities. Each day, it then forms a equal-weighted dollar-neutral portfolio of the 10 companies most likely to outperform and the 10 companies most likely to underperform.

```
public class KavoutCompositeFactorBundleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        AddUniverse(SelectAssets);
    }

    private IEnumerable<Symbol> SelectAssets(IEnumerable<Fundamental> fundamentals)
    {
        // Filter for the highly traded stocks for more informed data from frequent market activities, which may
translate to more accurate prediction
        // Factors scores are only available for the ones with fundamentals
        return (from f in fundamentals
                where f.HasFundamentalData
                orderby f.DollarVolume descending
                select f.Symbol).Take(100);
    }

    public override void OnData(Slice slice)
    {
        // Get the current data from the Kavout dataset.
        var points = slice.Get<KavoutCompositeFactorBundle>()
            // Drop factors for assets that have no price.
            .Where(kvp => Securities[kvp.Key.Underlying].Price != 0);
        //// Demonstrate how to iterate through the data and access its members:
        //foreach(var kvp in points)
        //{
        //    var datasetSymbol = kvp.Key;
        //    var factors = kvp.Value;
        //    Quit(
        //        $"{Time} -- " +
        //        $"Asset Symbol: {datasetSymbol.Underlying}; " +
        //        $"Growth factor: {factors.Growth}"
```

```
        //    );
        //}

        // Only rebalance when there are new Kavout factors.
        if (points.Count() == 0)
        {
            return;
        }

        // Long the stocks with highest factor scores, which indicate higher return from various factors
        // Short the ones with lowest factor scores for lower return estimates
        var sortedByScore = points.OrderBy(kvp => TotalScore(kvp.Value))
            .Select(kvp => kvp.Key.Underlying);
        var longSymbols = sortedByScore.TakeLast(10);
        var shortSymbols = sortedByScore.Take(10);

        // Invest in equal-size and dollar-neutral to evenly dissipate individual capital risk, avoid non-systematic
risk, and better margin
        var targets = new List<PortfolioTarget>();
        targets.AddRange(longSymbols.Select(symbol => new PortfolioTarget(symbol, 0.05m)));
        targets.AddRange(shortSymbols.Select(symbol => new PortfolioTarget(symbol, -0.05m)));

        SetHoldings(targets, true);
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach(dynamic security in changes.AddedSecurities)
        {
            // Requesting factor bundle data for trade signal generation
            security.kavoutSymbol = AddData<KavoutCompositeFactorBundle>(security.Symbol).Symbol;

            // Historical Data
            var history = History(security.kavoutSymbol, 60, Resolution.Daily);
        }
        foreach (dynamic security in changes.RemovedSecurities)
        {
            // Remove the factor bundle data for this asset when it leaves the universe.
            RemoveSecurity(security.kavoutSymbol);
        }
    }

    private decimal TotalScore(KavoutCompositeFactorBundle value)
    {
        // Return the total score to integrate overall likelihood to outcompete, take equal weighting for each factor
        return value.Growth + value.ValueFactor + value.Quality + value.Momentum + value.LowVolatility;
    }
}
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of the 100 most liquid US Equities. Each day, it then forms a equal-weighted dollar-neutral portfolio of the 10 companies most likely to outperform and the 10 companies most likely to underperform.

C#

```
public class KavoutCompositeFactorBundleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        AddUniverseSelection(new LiquidEquitiesUniverseSelectionModel());
        // Custom alpha model to emit insights based on factor bundle data
        AddAlpha(new KavoutCompositeFactorBundleAlphaModel());
        // Invest in equal-size and dollar-neutral to evenly dissipate individual capital risk, avoid non-systematic
risk, and better margin
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
        SetExecution(new ImmediateExecutionModel());
    }
}

public class LiquidEquitiesUniverseSelectionModel : FundamentalUniverseSelectionModel
{
```

```
    public override IEnumerable<Symbol> Select(QCAlgorithm algorithm, IEnumerable<Fundamental> fundamentals)
    {
        // Filter for the highly traded stocks for more informed data from frequent market activities, which may
translate to more accurate prediction
        // Factors scores are only available for the ones with fundamentals
        return (from f in fundamentals
                where f.HasFundamentalData
                orderby f.DollarVolume descending
                select f.Symbol).Take(100);
    }
}

public class KavoutCompositeFactorBundleAlphaModel: AlphaModel
{
    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        // Get the current data from the Kavout dataset.
        var points = slice.Get<KavoutCompositeFactorBundle>()
            // Drop factors for assets that have no price.
            .Where(kvp => algorithm.Securities[kvp.Key.Underlying].Price != 0);
        //// Demonstrate how to iterate through the data and access its members:
        //foreach(var kvp in points)
        //{
        //    var datasetSymbol = kvp.Key;
        //    var factors = kvp.Value;
        //    algorithm.Quit(
        //        $"{algorithm.Time} -- " +
        //        $"Asset Symbol: {datasetSymbol.Underlying}; " +
        //        $"Growth factor: {factors.Growth}"
        //    );
        //}

        // Only rebalance when there are new Kavout factors.
        if (points.Count() == 0)
        {
            return new List<Insight>();
        }

        // Long the stocks with highest factor scores, which indicate higher return from various factors
        // Short the ones with lowest factor scores for lower return estimates
        var sortedByScore = points.OrderBy(kvp => TotalScore(kvp.Value))
            .Select(kvp => kvp.Key.Underlying);
        var longSymbols = sortedByScore.TakeLast(10);
        var shortSymbols = sortedByScore.Take(10);

        var insights = new List<Insight>();
        insights.AddRange(longSymbols.Select(symbol => new Insight(symbol, Expiry.EndOfDay, InsightType.Price,
InsightDirection.Up)));
        insights.AddRange(shortSymbols.Select(symbol => new Insight(symbol, Expiry.EndOfDay, InsightType.Price,
InsightDirection.Down)));

        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach(dynamic security in changes.AddedSecurities)
        {
            // Requesting factor bundle data for trade signal generation
            security.kavoutSymbol = algorithm.AddData<KavoutCompositeFactorBundle>(security.Symbol).Symbol;
            // Historical Data
            var history = algorithm.History(security.kavoutSymbol, 10, Resolution.Daily);
        }
        foreach (dynamic security in changes.RemovedSecurities)
        {
            // Remove the factor bundle data for this asset when it leaves the universe.
            algorithm.RemoveSecurity(security.kavoutSymbol);
        }
    }

    private decimal TotalScore(KavoutCompositeFactorBundle value)
    {
        // Return the total score to integrate overall likelihood to outcompete, take equal weighting for each factor
        return value.Growth + value.ValueFactor + value.Quality + value.Momentum + value.LowVolatility;
    }
}
```

Data Point Attributes

The Composite Factor Bundle dataset provides `KavoutCompositeFactorBundle` objects, which have the following attributes:

Datasets

Nasdaq

" Nasdaq " was initially an acronym for the National Association of Securities Dealers Automated Quotations. It was founded in 1971 by the National Association of Securities Dealers (NASD), now known as the Financial Industry Regulatory Authority (FINRA), with the goal to provide financial services and operate stock exchanges.

On Dec. 4th, 2018, Nasdaq announced it had acquired Quandl, Inc. , a leading provider of alternative and core financial data. Quandl was founded by Tammer Kamel in 2012, with goal of making it easy for anyone to find and use high-quality data effectively in their professional decision-making. In 2021, Quandl was replaced by Nasdaq Data Link .

Data Link

Nasdaq

Data Link

Introduction

The Data Link dataset by Nasdaq, previously known as Quandl, is a collection of alternative data sets. It has indexed millions of time-series datasets from over 400 sources, which start in different years. This dataset is delivered on several frequencies, but the free data sets have often a daily frequency.

For more information about the Data Link dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

" Nasdaq " was initially an acronym for the National Association of Securities Dealers Automated Quotations. It was founded in 1971 by the National Association of Securities Dealers (NASD), now known as the Financial Industry Regulatory Authority (FINRA), with the goal to provide financial services and operate stock exchanges.

On Dec. 4th, 2018, Nasdaq announced it had acquired Quandl, Inc. , a leading provider of alternative and core financial data. Quandl was founded by Tammer Kamel in 2012, with goal of making it easy for anyone to find and use high-quality data effectively in their professional decision-making. In 2021, Quandl was replaced by Nasdaq Data Link .

Getting Started

The following snippet demonstrates how to request data from the Data Link dataset:

```C#
// For premium datasets, provide your API Key
// NasdaqDataLink.SetAuthCode(GetParameter("nasdaq-data-link-api-key"));

_bitcoinChainSymbol = AddData<NasdaqDataLink>("QDL/BCHAIN", Resolution.Daily).Symbol;
// This dataset has one data column ("Value")

_bitfinexExchangeRateSymbol = AddData<NasdaqCustomColumns>("QDL/BITFINEX", Resolution.Daily).Symbol;
// This dataset has multiple data columns. Create a subclass to set the default value column.

public class NasdaqCustomColumns : NasdaqDataLink
{
    // Select the column "mid"
    public NasdaqCustomColumns() : base(valueColumnName: "mid") { }
}
```

Our Nasdaq Data Link integration supports any dataset from Nasdaq that has a URL starting with `data.nasdaq.com/data/` . To import a dataset from Data Link, you need to get the dataset code. Follow these steps to get the dataset code:

1. Open the Data Link catalog .
2. *(Optional)* Use the filters on the left side bar to narrow down the catalog results.
3. Click one of the data products.
4. On the data product page, click one of the table names.
5. In the top-right corner of the dataset page, copy the Nasdaq Data Link Code.

Data Summary

The data summary depends on the specific Data Link dataset you use. Follow these steps to view the data summary of a dataset:

1. Open the Data Link catalog .
2. Click one of the data products.
3. On the data product page, click one of the table names.
4. View the data frequency and description in the left sidebar.
5. Above the chart, click `Table` to view the start date, end date, and data point attributes.

Backward Compatibility

QuantConnect/LEAN has supported Quandl since 2015. On January, 12, 2022, we moved the implementation from the LEAN GitHub repository to the Lean.DataSource.NasdaqDataLink GitHub repository . Through this transition, we maintained backward compatibility. All of the preceding code snippets work if you replace `NasdaqDataLink` with `Quandl` .

Requesting Data

To add Data Link data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm. If there is more than one value column in the Data Link dataset, to set the `Value` property of the data objects, create a sublcass of the `NasdaqDataLink` class and set its `ValueColumnName` property to the column name.

```
C#
public class NasdaqDataLinkDataAlgorithm : QCAlgorithm
{
    private Symbol _bitcoinChainSymbol , _bitfinexExchangeRateSymbol ;

    public override void Initialize()
    {
        SetStartDate(2022, 1, 1);
        SetEndDate(2022, 7, 1);
        SetCash(100000);

        // For premium datasets, provide your API Key
        // NasdaqDataLink.SetAuthCode(GetParameter("nasdaq-data-link-api-key"));

        _bitcoinChainSymbol = AddData<NasdaqDataLink>("QDL/BCHAIN", Resolution.Daily).Symbol;
        // This dataset has one data column ("Value")
        // Source : https://data.nasdaq.com/databases/BCHAIN

        _bitfinexExchangeRateSymbol = AddData<NasdaqCustomColumns>("QDL/BITFINEX", Resolution.Daily).Symbol;
        // This dataset has multiple data columns
        // Source: https://data.nasdaq.com/databases/BITFINEX
    }
}

public class NasdaqCustomColumns : NasdaqDataLink
{
    // Select the column "mid"
    public NasdaqCustomColumns() : base(valueColumnName: "mid") { }
}
```

Accessing Data

To get the current Data Link data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

To get the default value of the dataset, use the `Value` property. To get the value of a specific column in the dataset, call the `GetStorageDictionary` method and index the result with the column name.

```
public override void OnData(Slice slice)
{
    if (slice.TryGetValue(_bitcoinChainSymbol, out var point))
    {
        var value = point.Value;
        Log($"{_bitcoinChainSymbol} value at {slice.Time}: {value}");
    }

    if (slice.TryGetValue(_bitfinexExchangeRateSymbol, out var point))
    {
        var value = point.Value;
        var storageDictionary = point.GetStorageDictionary();
        var volume = storageDictionary["volume"];
        Log($"{_bitfinexExchangeRateSymbol} Bitfinex exchange rate at {slice.Time}: {value}");
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<NasdaqDataLink>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} value at {slice.Time}: {dataPoint.Value}");
    }

    foreach (var kvp in slice.Get<NasdaqCustomColumns>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} value at {slice.Time}: {dataPoint.Value}");
    }
}
```

Historical Data

The process to get historical Data Link data depends on your environment.

In Algorithms

To get historical Data Link data in an algorithm, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
var bitcoinChainHistory = History<NasdaqDataLink>(_bitcoinChainSymbol, 100, Resolution.Daily);
var bitfinexExchangeRateHistory = History<NasdaqDataLink>(_bitfinexExchangeRateSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

In Research Notebooks

To get historical Data Link data in the Research Environment, call the `Download` method with the data URL.

```
var data = qb.Download("https://data.nasdaq.com/api/v3/datatables/QDL/BCHAIN.csv?");
```

To receive a notification when you can use the `History` method in the Research Environment to get data from Data Link, subscribe to Lean.DataSource.NasdaqDataLink GitHub Issue #10 .

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```
C#

RemoveSecurity(_bitcoinChainSymbol);
RemoveSecurity(_bitfinexExchangeRateSymbol);
```

Example Applications

The Nasdaq Data Link sources allow you to explore different kinds of data in their database to develop trading strategies. Examples include the following strategies:

- Using alternative data to regress market regime/asset price.
- Backtesting exotic derivatives or private Equity investments.

Classic Algorithm Example

The following example algorithm subscribes to the Bitfinex Crypto Exchange Rate datasets. The algorithm takes a long position in Bitfinex BTCUSD.

```
C#

public class NasdaqImporterAlgorithm : QCAlgorithm
{
    private string nasdaqCode = "QDL/BITFINEX";
    private SimpleMovingAverage sma;

    public override void Initialize()
    {
        // Optional argument - personal token necessary for restricted dataset
        // NasdaqDataLink.SetAuthCode(this.GetParameter("nasdaq-data-link-api-key"));
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(25000);
        // Request nasdaq data link data to trade
        AddData<NasdaqCustomColumns>(nasdaqCode, Resolution.Daily, TimeZones.NewYork);
        // Create SMA indicator that feed with nasdaq data link data, which can be used to determine trend for trend
following trade
        sma = SMA(nasdaqCode, 14);
    }

    public override void OnData(Slice data)
    {
        // OnData event is the primary entry point for your algorithm. Each new data point will be pumped in here.
        if (!Portfolio.HoldStock)
        {
            SetHoldings(nasdaqCode, 1);
            Debug("Purchased " + nasdaqCode + " >> " + Time);
        }

        Plot(nasdaqCode, "PriceSMA", sma.Current.Value);
    }
}

// NasdaqDataLink often doesn't use close columns so need to tell LEAN which is the "value" column.
public class NasdaqCustomColumns : NasdaqDataLink
{
    // Custom nasdaq data type for setting customized value column name.
    // Value column is used for the primary trading calculations and charting.
    public NasdaqCustomColumns() : base("mid")
    {
        ValueColumnName = "mid";
    }
}
```

Framework Algorithm Example

The following example algorithm subscribes to the Bitcoin Chain dataset and Bitfinex Crypto Exchange Rate dataset. The

algorithm takes a long position in Bitfinex BTCUSD.

```C#
public class NasdaqDataLinkFrameworkAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        /// Requesting Data
        // For premium datasets, provide your API Key
        // NasdaqDataLink.SetAuthCode(GetParameter("nasdaq-data-link-api-key"));

        // Add nasdaq data link data as trading vehicle
        var bitcoinChainSymbol = AddData<NasdaqDataLink>("QDL/BCHAIN", Resolution.Daily).Symbol;

        /// Historical Data
        var history = History<NasdaqDataLink>(bitcoinChainSymbol, 1000, Resolution.Daily);
        Debug($"We got {history.Count()} items from our history request for {bitcoinChainSymbol} Nasdaq Data Link");

        // Custom alpha model that emit insights based on updated nasdaq data link data
        AddAlpha(new NasdaqDataLinkAlphaModel(this));

        // Invest equally to evenly dissipate the capital concentration risk
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel(time => null));
    }
}

public class NasdaqDataLinkAlphaModel : AlphaModel
{
    private Symbol _bitfinexExchangeRateSymbol;

    public NasdaqDataLinkAlphaModel(QCAlgorithm algorithm)
    {
        /// Requesting NASDAQ data link custom data
        _bitfinexExchangeRateSymbol = algorithm.AddData<NasdaqCustomColumns>("QDL/BITFINEX",
Resolution.Daily).Symbol;
        // This dataset has multiple data columns, so create a custom class to set the value column name

        /// Historical Data
        var history = algorithm.History<NasdaqCustomColumns>(_bitfinexExchangeRateSymbol, 100, Resolution.Daily);
        algorithm.Debug($"We got {history.Count()} items from our history request for {_bitfinexExchangeRateSymbol}
Nasdaq Data Link");
    }

    public override List<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        List<Insight> insights = new List<Insight>();

        /// Trade only based on updated NASDAQ data link data
        foreach (var kvp in slice.Get<NasdaqCustomColumns>())
        {
            var datasetSymbol = kvp.Key;
            if (datasetSymbol != _bitfinexExchangeRateSymbol || algorithm.Portfolio[datasetSymbol].Invested)
            {
                continue;
            }
            insights.Add(Insight.Price(datasetSymbol, TimeSpan.FromDays(6*22), InsightDirection.Up));
        }

        return insights;
    }
}

public class NasdaqCustomColumns : NasdaqDataLink
{
    // Select the column "mid".
    public NasdaqCustomColumns() : base(valueColumnName: "mid") { }
}
```

Datasets

Quiver Quantitative

Quiver Quantitative was founded by two college students in February 2020 with the goal of bridging the information gap between Wall Street and non-professional investors. Quiver allows retail investors to tap into the power of big data and have access to actionable, easy to interpret data that hasn't already been dissected by Wall Street.

CNBC Trading

Corporate Lobbying

Insider Trading

US Congress Trading

US Government Contracts

WallStreetBets

Quiver Quantitative

CNBC Trading

Introduction

The CNBC Trading dataset by Quiver Quantitative tracks the recommendations made by media personalities on CNBC and their historical performance. The data covers over 1,500 US Equities, starts in December 2020, and is delivered on a daily frequency. This dataset covers recommendations made on Mad Money, Halftime Report, and Fast Money.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the CNBC Trading dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Quiver Quantitative was founded by two college students in February 2020 with the goal of bridging the information gap between Wall Street and non-professional investors. Quiver allows retail investors to tap into the power of big data and have access to actionable, easy to interpret data that hasn't already been dissected by Wall Street.

Getting Started

The following snippet demonstrates how to request data from the CNBC Trading dataset:

```C#
_symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
_datasetSymbol = AddData<QuiverCNBCs>(_symbol).Symbol;

_universe = AddUniverse<QuiverCNBCsUniverse>(UniverseSelection);
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	December 25, 2020
Asset Coverage	1,515 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Requesting Data

To add CNBC Trading data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```
C#
public class QuiverCNBCDataAlgorithm: QCAlgorithm
{
    private Symbol _symbol, _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);
        _symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _datasetSymbol= AddData<QuiverCNBCs>(_symbol).Symbol;
    }
}
```

Accessing Data

To get the current CNBC Trading data, index the current `Slice` with the dataset `Symbol` . `Slice` objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var dataPoints = slice[_datasetSymbol];
        foreach (var dataPoint in dataPoints)
        {
            Log($"{_datasetSymbol} direction at {slice.Time}: {dataPoint.Direction}");
        }
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<QuiverCNBCs>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoints = kvp.Value;
        foreach(var dataPoint in dataPoints)
        {
            Log($"{datasetSymbol} direction at {slice.Time}: {dataPoint.Direction}");
        }
    }
}
```

Historical Data

To get historical CNBC Trading data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
var history = History<QuiverCNBCs>(_datasetSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on CNBC Trading data, call the `AddUniverse` method with the

`QuiverCNBCsUniverse` class and a selection function.

```C#
private Universe _universe;
public override void Initialize()
{
    _universe = AddUniverse<QuiverCNBCsUniverse>(altCoarse =>
    {
        var cnbcDataBySymbol = new Dictionary<Symbol, List<QuiverCNBCsUniverse>>();

        foreach (var datum in altCoarse.OfType<QuiverCNBCsUniverse>())
        {
            var symbol = datum.Symbol;

            if (!cnbcDataBySymbol.ContainsKey(symbol))
            {
                cnbcDataBySymbol.Add(symbol, new List<QuiverCNBCsUniverse>());
            }
            cnbcDataBySymbol[symbol].Add(datum);
        }

        // define our selection criteria
        return from kvp in cnbcDataBySymbol
            where kvp.Value.Where(x => x.Direction == OrderDirection.Buy) >= 3
            select kvp.Key;
    });
}
```

Universe History

You can get historical universe data in an algorithm and in the Research Environment.

Historical Universe Data in Algorithms

To get historical universe data in an algorithm, call the `History` method with the `Universe` object and the lookback period. If there is no data in the period you request, the history result is empty.

```C#
var universeHistory = History(universe, 30, Resolution.Daily);
foreach (var cnbcs in universeHistory)
{
    foreach (QuiverCNBCsUniverse cnbc in cnbcs)
    {
        Log($"{cnbc.Symbol} traders at {cnbc.EndTime}: {cnbc.Traders}");
    }
}
```

Historical Universe Data in Research

To get historical universe data in research, call the `UniverseHistory` method with the `Universe` object, a start date, and an end date. This method returns the filtered universe. If there is no data in the period you request, the history result is empty.

```C#
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var cnbcs in universeHistory)
{
    foreach (QuiverCNBCsUniverse cnbc in cnbcs)
    {
        Console.WriteLine($"{cnbc.Symbol} traders at {cnbc.EndTime}: {cnbc.Traders}");
    }
}
```

You can call the `History` method in Research.

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```
C#
RemoveSecurity(_datasetSymbol);
```

If you subscribe to CNBC Trading data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Quiver Quantitative CNBC Trading dataset enables you to create strategies using the latest recommendations made by media personalities on CNBC. Examples include the following strategies:

- Taking short positions in securities that were mentioned by Jim Cramer (CNBC commentator) in the last week
- Trading securities that were most/least discussed across CNBC programs over the last year

Classic Algorithm Example

The following example algorithm buys Apple stock if the net recommendation of media personalities on CNBC for Apple is positive. Otherwise, it holds cash.

```
C#
public class QuiverCNBCsAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _datasetSymbol;
    private Dictionary<OrderDirection, int> _intByOrderDirection;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        _symbol = AddEquity("AAPL").Symbol;
        // Subscribe to CNBC data for AAPL to generate trade signal
        _datasetSymbol = AddData<QuiverCNBCs>(_symbol).Symbol;
        // Define a mapping from OrderDirection to int.
        _intByOrderDirection = new () {
            {OrderDirection.Buy, 1},
            {OrderDirection.Hold, 0},
            {OrderDirection.Sell, -1}
        };

        // history request
        var history = History<QuiverCNBCs>(new[] {_datasetSymbol}, 10, Resolution.Daily);
        Debug($"We got {history.Count()} items from historical data request of {_datasetSymbol}.");
    }

    public override void OnData(Slice slice)
    {
        foreach (var kvp in slice.Get<QuiverCNBCs>())
        {
            // Using mean prediction from CNBC analysts to be the trade signal
            // If the average CNBC insight is upward movement, invest AAPL
            if (kvp.Value.Average(x => _intByOrderDirection[(x as QuiverCNBC).Direction]) > 0)
            {
                SetHoldings(_symbol, 1);
            }
            else
            {
                SetHoldings(_symbol, 0);
            }
        }
    }
}
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of US Equities that have at least 3 positive opinions from CNBC

sources. Each day, it then forms a equal-weighted portfolio with all the securities in the universe.

```C#
public class QuiverCNBCsDataAlgorithm : QCAlgorithm
{
    private Dictionary<Symbol, Symbol> _datasetSymbolBySymbol = new();
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Filter universe based on CNBC data
        AddUniverse<QuiverCNBCsUniverse>(data =>
        {
            var cnbcDataBySymbol = new Dictionary<Symbol, List<QuiverCNBCsUniverse>>();

            foreach (var datum in data.OfType<QuiverCNBCsUniverse>())
            {
                var symbol = datum.Symbol;

                if (!cnbcDataBySymbol.ContainsKey(symbol))
                {
                    cnbcDataBySymbol.Add(symbol, new List<QuiverCNBCsUniverse>());
                }
                cnbcDataBySymbol[symbol].Add(datum);
            }

            // Select the stocks with at least 3 CNBC analysts to suggest buy, reassuring the signal
            return from kvp in cnbcDataBySymbol
                where kvp.Value.Where(x => x.Direction == OrderDirection.Buy).Count() >= 3
                select kvp.Key;
        });

        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(1)));

        // Invest equally to evenly dissipate the capital concentration risk
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            // Requesting CNBC Data
            var symbol = security.Symbol;
            var datasetSymbol = AddData<QuiverCNBCs>(symbol).Symbol;
            _datasetSymbolBySymbol.Add(symbol, datasetSymbol);

            // History request
            var history = History<QuiverCNBCs>(datasetSymbol, 10, Resolution.Daily);
            Debug($"We get {history.Count()} items in historical data of {datasetSymbol}");
        }

        foreach (var security in changes.RemovedSecurities)
        {
            var symbol = security.Symbol;
            if (_datasetSymbolBySymbol.ContainsKey(symbol))
            {
                // Remove subscription of CNBC data to release computation resources
                _datasetSymbolBySymbol.Remove(symbol, out var datasetSymbol);
                RemoveSecurity(datasetSymbol);
            }
        }
    }
}
```

Research Example

The following example lists US Equities mentioned by Jim Cramer.

C#

```
#r "../QuantConnect.DataSource.QuiverCNBC.dll"
using QuantConnect.DataSource;

var qb = new QuantBook();

// Requesting data
var aapl = qb.AddEquity("AAPL", Resolution.Daily).Symbol;
var symbol = qb.AddData<QuiverCNBCs>(aapl).Symbol;

// Historical data
var history = qb.History<QuiverCNBCs>(symbol, 60, Resolution.Daily);
foreach (var cnbcs in history)
{
    foreach (QuiverCNBC cnbc in cnbcs)
    {
        Console.WriteLine($"{cnbc.Symbol} traders at {cnbc.EndTime}: {cnbc.Traders}");
    }
}

// Add Universe Selection
IEnumerable<Symbol> UniverseSelection(IEnumerable<BaseData> altCoarse)
{
    return from d in altCoarse.OfType<QuiverCNBCsUniverse>()
        where d.Traders.Contains("Cramer") select d.Symbol;
}
var universe = qb.AddUniverse<QuiverCNBCsUniverse<(UniverseSelection);

// Historical Universe data
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-60), qb.Time);
foreach (var cnbcs in universeHistory)
{
    foreach (QuiverCNBCsUniverse cnbc in cnbcs)
    {
        Console.WriteLine($"{cnbc.Symbol} traders at {cnbc.EndTime}: {cnbc.Traders}");
    }
}
```

Data Point Attributes

The Quiver Quantitative CNBC Trading dataset provides `QuiverCNBCs` , `QuiverCNBC` , and `QuiverCNBCsUniverse` objects.

QuiverCNBCs

`QuiverCNBCs` objects have the following attributes:

QuiverCNBC

`QuiverCNBC` objects have the following attributes:

QuiverCNBCsUniverse

`QuiverCNBCsUniverse` objects have the following attributes:

Quiver Quantitative

Corporate Lobbying

Introduction

The Corporate Lobbying dataset by Quiver Quantitative tracks the lobbying activity of US Equities. The Lobbying Disclosure Act of 1995 requires lobbyists in the United States to disclose information about their activities, such as their clients, which issues they are lobbying on, and how much they are being paid. Quiver Quantiative scrapes this data and maps it to stock tickers to track which companies are spending money for legislative influence.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Corporate Lobbying dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Quiver Quantitative was founded by two college students in February 2020 with the goal of bridging the information gap between Wall Street and non-professional investors. Quiver allows retail investors to tap into the power of big data and have access to actionable, easy to interpret data that hasn't already been dissected by Wall Street.

Getting Started

The following snippet demonstrates how to request data from the Corporate Lobbying dataset:

```C#
_symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
_datasetSymbol = AddData<QuiverLobbyings>(_symbol).Symbol;

_universe = AddUniverse<QuiverLobbyingUniverse>(UniverseSelectionFilter);
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 4, 1999
Asset Coverage	1,418 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Requesting Data

To add Corporate Lobbying data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```
C#
public class QuiverLobbyingDataAlgorithm: QCAlgorithm
{
    private Symbol _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);
        var symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _datasetSymbol= AddData<QuiverLobbyings>(symbol).Symbol;
    }
}
```

Accessing Data

To get the current Corporate Lobbying data, index the current `Slice` with the dataset `Symbol` . `Slice` objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var dataPoints = slice[_datasetSymbol];
        foreach (var dataPoint in dataPoints)
        {
            Log($"{_datasetSymbol} amount at {slice.Time}: {dataPoint.Amount}");
        }
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<QuiverLobbyings>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoints = kvp.Value;
        foreach(var dataPoint in dataPoints)
        {
            Log($"{datasetSymbol} amount at {slice.Time}: {dataPoint.Amount}");
        }
    }
}
```

Historical Data

To get historical Corporate Lobbying data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
var history = History<QuiverLobbyings>(_datasetSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on Corporate Lobbying data, call the `AddUniverse` method with the

`QuiverLobbyingUniverse` class and a selection function.

```
C#
private Universe _universe;
public override void Initialize()
{
    _universe = AddUniverse<QuiverLobbyingUniverse>("QuiverLobbyingUniverse", Resolution.Daily, altCoarse =>
    {
        var lobbyDataBySymbol = new Dictionary<Symbol, List<QuiverLobbyingUniverse>>();

        foreach (var datum in altCoarse.OfType<QuiverLobbyingUniverse>())
        {
            var symbol = datum.Symbol;

            if (!lobbyDataBySymbol.ContainsKey(symbol))
            {
                lobbyDataBySymbol.Add(symbol, new List<QuiverLobbyingUniverse>());
            }
            lobbyDataBySymbol[symbol].Add(datum);
        }

        return from kvp in lobbyDataBySymbol
               where kvp.Value.Sum(x => x.Amount) >= 100000
               select kvp.Key;
    })
};
```

Universe History

You can get historical universe data in an algorithm and in the Research Environment.

Historical Universe Data in Algorithms

To get historical universe data in an algorithm, call the `History` method with the `Universe` object and the lookback period. If there is no data in the period you request, the history result is empty.

```
C#
var universeHistory = History(universe, 30, Resolution.Daily);
foreach (var lobbyings in universeHistory)
{
    foreach (QuiverLobbyingUniverse lobbying in lobbyings)
    {
        Log($"{lobbying.Symbol} issue at {lobbying.EndTime}: {lobbying.Issue}");
    }
}
```

Historical Universe Data in Research

To get historical universe data in research, call the `UniverseHistory` method with the `Universe` object, a start date, and an end date. This method returns the filtered universe. If there is no data in the period you request, the history result is empty.

```
C#
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var lobbyings in universeHistory)
{
    foreach (QuiverLobbyingUniverse lobbying in lobbyings)
    {
        Consolte.WriteLine($"{lobbying.Symbol} issue at {lobbying.EndTime}: {lobbying.Issue}");
    }
}
```

You can call the `History` method in Research.

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```
C#
RemoveSecurity(_datasetSymbol);
```

If you subscribe to Corporate Lobbying data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Corporate Lobbying dataset enables you to create strategies using the latest information on lobbying activity. Examples include the following strategies:

- Trading securities that have spent the most on lobbying over the last quarter
- Trading securities that have had the biggest change in lobbying spend for privacy legislation over the last year

Classic Algorithm Example

The following example algorithm tracks the lobbying activity of Apple. When the company initiates a lobbying activity worth more than $50K, the algorithm buys Apple stock. When the company initiates a lobbying activity worth less than $10K, the algorithm short sells Apple stock.

```
C#
public class QuiverLobbyingAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        _symbol = AddEquity("AAPL").Symbol;
        // Subscribe to lobbying data for AAPL to generate trade signal
        _datasetSymbol = AddData<QuiverLobbyings>(_symbol).Symbol;

        // history request
        var history = History<QuiverLobbyings>(new[] {_datasetSymbol}, 10, Resolution.Daily);
        Debug($"We got {history.Count()} items from historical data request of {_datasetSymbol}.");
    }

    public override void OnData(Slice slice)
    {
        // Trade only base on lobbying data
        foreach (var kvp in slice.Get<QuiverLobbyings>())
        {
            var lobbyings = kvp.Value;
            // Buy if over 50000 lobbying amount, suggesting a favored political prospect and sentiment
            if (lobbyings.Any(lobbying => ((QuiverLobbying) lobbying).Amount >= 50000m))
            {
                SetHoldings(_symbol, 1);
            }
            // Sell if below 10000 lobbying amount, suggesting a less favorable political prospect and sentiment
            else if (lobbyings.Any(lobbying => ((QuiverLobbying) lobbying).Amount <= 10000m))
            {
                SetHoldings(_symbol, -1);
            }
        }
    }
}
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of US Equities that have at least $100K worth of new lobbying activity. Each day, it then forms an equal-weighted portfolio with all of the securities in the universe.

C#

```
public class QuiverLobbyingDataAlgorithm : QCAlgorithm
{
    // To hold the lobbying dataset symbol for managing subscription
    private Dictionary<Symbol, Symbol> _datasetSymbolBySymbol = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Filter universe based on the lobbying data
        AddUniverse<QuiverLobbyingUniverse>( data =>
        {
            var lobbyDataBySymbol = new Dictionary<Symbol, List<QuiverLobbyingUniverse>>();

            foreach (var datum in data.OfType<QuiverLobbyingUniverse>())
            {
                var symbol = datum.Symbol;

                if (!lobbyDataBySymbol.ContainsKey(symbol))
                {
                    lobbyDataBySymbol.Add(symbol, new List<QuiverLobbyingUniverse>());
                }
                lobbyDataBySymbol[symbol].Add(datum);
            }

            // Select and invest all stocks with lobbying amount above 100000, suggesting a more favorable political
prospect and sentiment
            return from kvp in lobbyDataBySymbol
                where kvp.Value.Sum(x => x.Amount) >= 100000m
                select kvp.Key;
        });

        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(1)));

        // Invest equally to evenly dissipate the capital concentration risk
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            // Requesting lobbying data for trading
            var symbol = security.Symbol;
            var datasetSymbol = AddData<QuiverLobbyings>(symbol).Symbol;
            _datasetSymbolBySymbol .Add(symbol, datasetSymbol);
            // History request
            var history = History<QuiverLobbyings>(datasetSymbol, 10, Resolution.Daily);
        }

        foreach (var security in changes.RemovedSecurities)
        {
            var symbol = security.Symbol;
            if (_datasetSymbolBySymbol .ContainsKey(symbol))
            {
                // Remove lobbying data subscription to release computation resources
                _datasetSymbolBySymbol .Remove(symbol, out var datasetSymbol);
                RemoveSecurity(datasetSymbol);
            }
        }
    }
}
```

Research Example

The following example lists US Equities that have brought health related issues.

```C#
#r "../QuantConnect.DataSource.QuiverLobbying.dll"
using QuantConnect.DataSource;

var qb = new QuantBook();

// Requesting data
var aapl = qb.AddEquity("AAPL", Resolution.Daily).Symbol;
var symbol = qb.AddData<QuiverLobbyings>(aapl).Symbol;

// Historical data
var history = qb.History<QuiverLobbyings>(symbol, 60, Resolution.Daily);
foreach (var lobbyings in history)
{
    foreach (QuiverLobbying lobbying in lobbyings)
    {
        Console.WriteLine($"{lobbying.Symbol} issue at {lobbying.EndTime}: {lobbying.Issue}");
    }
}

// Add Universe Selection
IEnumerable<Symbol> UniverseSelection(IEnumerable<BaseData> altCoarse)
{
    return from d in altCoarse.OfType<QuiverLobbyingUniverse>()
        where d.Issue.ToLower().Contains("health") select d.Symbol;
}
var universe = qb.AddUniverse(UniverseSelection);

// Historical Universe data
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-60), qb.Time);
foreach (var lobbyings in universeHistory)
{
    foreach (QuiverLobbyingUniverse lobbying in lobbyings)
    {
        Console.WriteLine($"{lobbying.Symbol} issue at {lobbying.EndTime}: {lobbying.Issue}");
    }
}
```

Data Point Attributes

The Quiver Quantitative Corporate Lobbying dataset provides `QuiverLobbyings` , `QuiverLobbying` , and `QuiverLobbyingUniverse` objects.

QuiverLobbyings

`QuiverLobbyings` objects have the following attributes:

QuiverLobbying

`QuiverLobbying` objects have the following attributes:

QuiverLobbyingUniverse

`QuiverLobbyingUniverse` objects have the following attributes:

Quiver Quantitative

Insider Trading

Introduction

Corporate insiders are required to disclose purchases or sales of their own stock within two business days of when they occur. Using these disclosures, we collect data on insider trading activity, which can give hints on whether executives are bullish or bearish on their own companies. Here is a blog that we did on this dataset: https://www.quiverquant.com/blog/081121

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Insider Trading dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Quiver Quantitative was founded by two college students in February 2020 with the goal of bridging the information gap between Wall Street and non-professional investors. Quiver allows retail investors to tap into the power of big data and have access to actionable, easy to interpret data that hasn't already been dissected by Wall Street.

Getting Started

The following snippet demonstrates how to request data from the Insider Trading dataset:

```
C#
_symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
_datasetSymbol = AddData<QuiverInsiderTrading>(aapl).Symbol;
_universe = AddUniverse<QuiverInsiderTradingUniverse>(UniverseSelection);
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	25 April 2014
Asset Coverage	4994 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Requesting Data

To add Insider Trading data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```C#
public class QuiverInsiderTradingDataAlgorithm: QCAlgorithm
{
    private Symbol _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);
        var symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _datasetSymbol= AddData<QuiverInsiderTrading>(symbol).Symbol;
    }
}
```

Accessing Data

To get the current Insider Trading data, index the current `Slice` with the dataset `Symbol` . `Slice` objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var dataPoints = slice[_datasetSymbol];
        foreach (QuiverInsiderTrading dataPoint in dataPoints)
        {
            Log($"{_datasetSymbol} shares at {slice.Time}: {dataPoint.Shares}");
        }
    }
}
```

Note that `dataPoints` is a list of `QuiverInsiderTrading` object since we can have several trades per symbol in a day. You need to iterate `dataPoints` to access the data.

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<QuiverInsiderTrading>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoints = kvp.Value;
        foreach(QuiverInsiderTrading dataPoint in dataPoints)
        {
            Log($"{datasetSymbol} shares at {slice.Time}: {dataPoint.Shares}");
        }
    }
}
```

Historical Data

To get historical Insider Trading data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
var history = History<QuiverInsiderTrading>(_datasetSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on Insider Trading data, call the `AddUniverse` method with the `QuiverInsiderTradingUniverse` class and a selection function.

```
C#
public class InsiderTradingUniverseAlgorithm : QCAlgorithm
{
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2023, 1, 1);
        _universe = AddUniverse<QuiverInsiderTradingUniverse>(SelectAssets);
    }

    private IEnumerable<Symbol> SelectAssets(IEnumerable<BaseData> altCoarse)
    {
        var dollarVolumeBySymbol = new Dictionary<Symbol, decimal?>();
        foreach (QuiverInsiderTradingUniverse data in altCoarse)
        {
            if (data.PricePerShare == 0m || data.PricePerShare == null)
            {
                continue;
            }
            if (!dollarVolumeBySymbol.ContainsKey(data.Symbol))
            {
                dollarVolumeBySymbol[data.Symbol] = 0;
            }
            dollarVolumeBySymbol[data.Symbol] += data.Shares * data.PricePerShare;
        }
        return dollarVolumeBySymbol
            .OrderByDescending(kvp => kvp.Value)
            .Take(10)
            .Select(kvp => kvp.Key);
    }
}
```

Universe History

You can get historical universe data in an algorithm and in the Research Environment.

Historical Universe Data in Algorithms

To get historical universe data in an algorithm, call the `History` method with the `Universe` object and the lookback period. If there is no data in the period you request, the history result is empty.

```
C#
var universeHistory = History(universe, 30, Resolution.Daily);
foreach (var insiders in universeHistory)
{
    foreach (QuiverInsiderTradingUniverse insider in insiders)
    {
        Log($"{insider.Symbol} volume at {insider.EndTime}: {insider.Shares * insider.PricePerShare}");
    }
}}
```

Historical Universe Data in Research

To get historical universe data in research, call the `UniverseHistory` method with the `Universe` object, a start date, and an end date. This method returns the filtered universe. If there is no data in the period you request, the history result is empty.

```C#
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var insiders in universeHistory)
{
    foreach (QuiverInsiderTradingUniverse insider in insiders)
    {
        Console.WriteLine($"{insider.Symbol} volume at {insider.EndTime}: {insider.Shares * insider.PricePerShare}");
    }
}
```

You can call the `History` method in Research.

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_datasetSymbol);
```

If you subscribe to Insider Trading data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Quiver Quantitative Insider Trading dataset enables researchers to create strategies using the latest information on insider trading activity. Examples include:

- Taking a short position in securities that have had the most insider selling over the last 5 days
- Buying any security that has had over $100,000 worth of shares purchased by insiders in the last month

Classic Algorithm Example

In this example, we would 100% long AAPL if there was any insider buying event. Otherwise, we liquidate it.

C#

```
public class QuiverInsiderTradingAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        _symbol = AddEquity("AAPL").Symbol;
        // Subscribe to insider trade data for AAPL to generate trade signal
        _datasetSymbol = AddData<QuiverInsiderTrading>(_symbol).Symbol;

        // history request
        var history = History<QuiverInsiderTrading>(new[] {_datasetSymbol}, 10, Resolution.Daily);
        Debug($"We got {history.Count()} items from historical data request of {_datasetSymbol}.");
    }

    public override void OnData(Slice slice)
    {
        // Trade only base on insider trade data
        foreach (var kvp in slice.Get<QuiverInsiderTrading>())
        {
            foreach (QuiverInsiderTrading insiderTrade in kvp.Value)
            {
                // Any buy insider trade will result in buying, assuming insider have confidence in stock price with
more informed information and projection
                if (insiderTrade.Shares > 0)
                {
                    SetHoldings(_symbol, 1);
                }
                // Any sell insider trade will result in liquidation, assuming insiders believe the stock price has
reached maximum or poor future confidence
                else
                {
                    Liquidate(_symbol);
                }
            }
        }
    }
}
```

Framework Algorithm Example

In this example, we long a equal-sized portfolio with equities that having insider trading for over $100000 USD in the previous day.

C#

```
public class QuiverInsiderTradingDataAlgorithm : QCAlgorithm
{
    // To hold the insider trade dataset symbol for managing subscription
    private Dictionary<Symbol, Symbol> _dataSymbols = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Filter universe using insider trade data
        AddUniverse<QuiverInsiderTradingUniverse>(data =>
        {
            var symbolData = new Dictionary<Symbol, List<QuiverInsiderTradingUniverse>>();

            foreach (var datum in data.OfType<QuiverInsiderTradingUniverse>())
            {
                var symbol = datum.Symbol;

                if (!symbolData.ContainsKey(symbol))
                {
                    symbolData.Add(symbol, new List<QuiverInsiderTradingUniverse>());
                }
                symbolData[symbol].Add(datum);
            }

            // Select the ones with insider trading dollar amount above 100k, assuming insider have confidence in
stock price with more informed information and projection
            return from kvp in symbolData
                where kvp.Value.Sum(x => x.Shares * x.PricePerShare) >= 100000
                select kvp.Key;
        });

        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(1)));

        // Invest equally to evenly dissipate the capital concentration risk
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            // Requesting insider trade data for trading
            var symbol = security.Symbol;
            var datasetSymbol = AddData<QuiverInsiderTrading>(symbol).Symbol;
            _dataSymbols.Add(symbol, datasetSymbol);
            // History request
            var history = History<QuiverInsiderTrading>(datasetSymbol, 10, Resolution.Daily);
        }

        foreach (var security in changes.RemovedSecurities)
        {
            var symbol = security.Symbol;
            if (_dataSymbols.ContainsKey(symbol))
            {
                // Remove insider trade data subscription to release computation resources
                _dataSymbols.Remove(symbol, out var datasetSymbol);
                RemoveSecurity(datasetSymbol);
            }
        }
    }
}
```

Research Example

The following example lists US Equities having insider trading for over $10,000,000.

C#

```
#r "../QuantConnect.DataSource.QuiverInsiderTrading.dll"
using QuantConnect.DataSource;

var qb = new QuantBook();

// Requesting data
var aapl = qb.AddEquity("AAPL", Resolution.Daily).Symbol;
var symbol = qb.AddData<QuiverInsiderTrading>(aapl).Symbol;

// Historical data
var history = qb.History<QuiverInsiderTrading>(symbol, 60, Resolution.Daily);
foreach (var insiders in history)
{
    foreach (QuiverInsiderTrading insider in insiders)
    {
        Console.WriteLine($"{insider.Symbol} shares at {insider.EndTime}: {insider.Shares}");
    }
}

// Add Universe Selection
IEnumerable<Symbol> UniverseSelection(IEnumerable<BaseData> altCoarse)
{
    return from d in altCoarse.OfType<QuiverInsiderTradingUniverse>()
        where d.Shares * d.PricePerShare >= 10000000 select d.Symbol;
}
var universe = qb.AddUniverse<QuiverInsiderTradingUniverse>(UniverseSelection);

// Historical Universe data
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-5), qb.Time);
foreach (var insiders in universeHistory)
{
    foreach (QuiverInsiderTradingUniverse insider in insiders)
    {
        Console.WriteLine($"{insider.Symbol} volume at {insider.EndTime}: {insider.Shares * insider.PricePerShare}");
    }
}
```

Data Point Attributes

The Insider Trading dataset provides `QuiverInsiderTrading` and `QuiverInsiderTradingUniverse` object.

QuiverInsiderTrading

`QuiverInsiderTrading` objects have the following attributes:

QuiverInsiderTradingUniverse

`QuiverInsiderTradingUniverse` objects have the following attributes:

Quiver Quantitative

US Congress Trading

Introduction

The US Congress Trading dataset by Quiver Quantitative tracks US Equity trades made by members of Congress in the Senate and the House of Representatives. The data covers 1,800 US Equities, starts in January 2016, and is delivered on a daily frequency. This dataset is created by scraping SEC reports.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the US Congress Trading dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Quiver Quantitative was founded by two college students in February 2020 with the goal of bridging the information gap between Wall Street and non-professional investors. Quiver allows retail investors to tap into the power of big data and have access to actionable, easy to interpret data that hasn't already been dissected by Wall Street.

Getting Started

The following snippet demonstrates how to request data from the US Congress Trading dataset:

```C#
_symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
_datasetSymbol = AddData<QuiverCongress>(_symbol).Symbol;

_universe = AddUniverse<QuiverQuantCongresssUniverse>(UniverseSelection);
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2016
Asset Coverage	1,800 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Requesting Data

To add US Congress Trading data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```
C#
public class QuiverCongressDataAlgorithm : QCAlgorithm
{
    private Symbol _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);
        var symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _datasetSymbol = AddData<QuiverCongress>(symbol).Symbol;
    }
}
```

Accessing Data

To get the current US Congress Trading data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var dataPoints = slice[_datasetSymbol];
        foreach (var dataPoint in dataPoints)
        {
            Log($"{_datasetSymbol} transaction amount at {slice.Time}: {dataPoint.Amount}");
        }
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<QuiverCongress>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} transaction amount at {slice.Time}: {dataPoint.Amount}");
    }
}
```

Historical Data

To get historical US Congress Trading data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
var history = History<QuiverCongress>(_datasetSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on US Congress Trading data, call the `AddUniverse` method with the `QuiverQuantCongressUniverse` class and a selection function.

```
C#
private Universe _universe;
public override void Initialize()
{
    _universe = AddUniverse<QuiverQuantCongresssUniverse>(altCoarse =>
    {
        return from d in altCoarse.OfType<QuiverCongressDataPoint>()
            where d.Amount > 200000 && d.Transaction == OrderDirection.Buy
            select d.Symbol;
    })
};
```

For more information about dynamic universes, see Universes .

Universe History

You can get historical universe data in an algorithm and in the Research Environment.

Historical Universe Data in Algorithms

To get historical universe data in an algorithm, call the `History` method with the `Universe` object and the lookback period. If there is no data in the period you request, the history result is empty.

```
C#
var universeHistory = History(_universe, 30, Resolution.Daily);
foreach (var trade in universeHistory.SelectMany(x => x).OfType<QuiverCongressDataPoint>())
{
    Log($"{trade.Symbol} amount at {trade.EndTime}: {trade.Amount} {trade.Representative}");
}
```

Historical Universe Data in Research

To get historical universe data in research, call the `UniverseHistory` method with the `Universe` object, a start date, and an end date. This method returns the filtered universe. If there is no data in the period you request, the history result is empty.

```
C#
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var trade in universeHistory.SelectMany(x => x).OfType<QuiverCongressDataPoint>())
{
    Console.WriteLine($"{trade.Symbol} amount at {trade.EndTime}: {trade.Amount} {trade.Representative}");
}
```

You can call the `History` method in Research.

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```
C#
RemoveSecurity(_datasetSymbol);
```

If you subscribe to US Congress Trading data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The US Congress Trading dataset enables you to take immediate action on trades made by informed Members of Congress. Examples include the following strategies:

- Following the trades of specific representatives on the premise that the representatives are more informed
- Assigning a long/short-bias to securities on a daily frequency based on how Members of Congress are trading them

Classic Algorithm Example

The following example algorithm follows the net bias of the trades made by Members of Congress each day. When Members of Congress are net buyers, the algorithm buys. When they are net sellers, the algorithm short sells.

```
C#
public class QuiverCongressDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Requesting data per underlying data to subscribe to updated congress members' trade information
        var aapl = AddEquity("AAPL", Resolution.Daily).Symbol;
        var quiverCongressSymbol = AddData<QuiverCongress>(aapl).Symbol;

        // Historical data
        var history = History<QuiverCongress>(quiverCongressSymbol, 60, Resolution.Daily);
        Debug($"We got {history.Count()} items from our history request");
    }


    public override void OnData(Slice slice)
    {
        var congressBySymbol = slice.Get<QuiverCongress>();

        // Determine net direction of Congress trades for each security to estimate the sentiment direction due to
political factor
        var netQuantityBySymbol = new Dictionary<Symbol, decimal>();
        foreach (var (s, points) in congressBySymbol)
        {
            var symbol = s.Underlying;
            if (!netQuantityBySymbol.ContainsKey(symbol))
            {
                netQuantityBySymbol[symbol] = 0m;
            }
            // Voting weight is by order size
            foreach(QuiverCongressDataPoint point in points)
            {
                netQuantityBySymbol[symbol] += (point.Transaction == OrderDirection.Buy ? 1 : -1) * (point.Amount ??
0m);
            }
        }

        foreach (var (symbol, netQuantity) in netQuantityBySymbol)
        {
            if (netQuantity == 0)
            {
                Liquidate(symbol);
                continue;
            }
            // Buy when Congress members have bought, short otherwise
            SetHoldings(symbol, netQuantity > 0 ? 1 : -1);
        }
    }
}
```

Framework Algorithm Example

The following example algorithm follows the net bias of the trades made by Members of Congress each day. When Members of Congress are net buyers, the algorithm buys. When they are net sellers, the algorithm short sells.

```
C#
public class QuiverCongressDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
```

```
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        // Filter according to QuiverCongress data
        AddUniverse<QuiverQuantCongressUniverse>("QuiverQuantCongresssUniverse", Resolution.Daily, altCoarse =>
        {
            // Only include the ones with large size buy, since they are estimated to be materially confident to go
up
            return from d in altCoarse.OfType<QuiverCongressDataPoint>()
                where d.Amount > 10000 && d.Transaction == OrderDirection.Buy
                select d.Symbol;
        });
        AddAlpha(new CongressAlphaModel());
        SetPortfolioConstruction(new InsightWeightingPortfolioConstructionModel(time => null));
        AddRiskManagement(new NullRiskManagementModel());
        SetExecution(new ImmediateExecutionModel());
    }
}

public class CongressAlphaModel : AlphaModel
{
    private Dictionary<Symbol, SymbolData> _symbolDataBySymbol = new ();
    private Dictionary<Symbol, Insight> _insightBySymbol = new ();

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var congressBySymbol = slice.Get<QuiverCongress>();
        if (congressBySymbol.Count > 0)
        {
            _insightBySymbol.Clear();
            // Determine net direction of Congress trades for each security to estimate the sentiment direction due
to political factor
            var netQuantityBySymbol = new Dictionary<Symbol, decimal>();
            foreach (var (s, points) in congressBySymbol)
            {
                var symbol = s.Underlying;
                if (!netQuantityBySymbol.ContainsKey(symbol))
                {
                    netQuantityBySymbol[symbol] = 0m;
                }
                // Voting weight is by order size
                foreach(QuiverCongressDataPoint point in points)
                {
                    netQuantityBySymbol[symbol] += (point.Transaction == OrderDirection.Buy ? 1 : -1) *
(point.Amount ?? 0m);
                }
            }

            foreach (var (symbol, netQuantity) in netQuantityBySymbol)
            {
                // Buy when Congress members have bought, as they may have advance information to be confident to the
return direction
                InsightDirection direction;
                if (netQuantity > 0 && !algorithm.Portfolio[symbol].IsLong)
                {
                    direction = InsightDirection.Up;
                }
                // Short sell when Congress members have sold
                else if (netQuantity < 0 && !algorithm.Portfolio[symbol].IsShort)
                {
                    direction = InsightDirection.Down;
                }
                else
                {
                    continue;
                }
                _insightBySymbol[symbol] = Insight.Price(symbol, TimeSpan.FromDays(7), direction, weight: 0.5);
            }
        }
        return _insightBySymbol.Keys
            .ToList() // snapshot keys
            .Where(symbol => algorithm.Securities[symbol].Price != 0m)
            .Select(symbol => {
                var val = _insightBySymbol[symbol]; // capture value
                _insightBySymbol.Remove(symbol);    // remove
                return val;
            })
            .ToList();
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
```

```
    {
        foreach (var security in changes.AddedSecurities)
        {
            var symbol = security.Symbol;
            _symbolDataBySymbol.Add(symbol, new SymbolData(algorithm, symbol));
        }

        foreach (var security in changes.RemovedSecurities)
        {
            var symbol = security.Symbol;
            if (_symbolDataBySymbol.ContainsKey(symbol))
            {
                _symbolDataBySymbol[symbol].dispose();
                _symbolDataBySymbol.Remove(symbol);
            }
        }
    }
}

public class SymbolData
{
    private Symbol _quiverCongressSymbol;
    private QCAlgorithm _algorithm;

    public SymbolData(QCAlgorithm algorithm, Symbol symbol)
    {
        _algorithm = algorithm;

        // Requesting data per underlying data to subscribe to updated congress members' trade information
        _quiverCongressSymbol = algorithm.AddData<QuiverCongress>(symbol).Symbol;

        // Historical data
        var history = algorithm.History<QuiverCongress>(_quiverCongressSymbol, 60, Resolution.Daily);
        algorithm.Debug($"We got {history.Count()} items from our history request for {symbol} Quiver Congress
data");
    }

    public void dispose()
    {
        // Unsubscribe from Quiver Congress feed for this security to release computational resources
        _algorithm.RemoveSecurity(_quiverCongressSymbol);
    }
}
```

Research Example

The following example lists trades made by Members of Congress daily as buyers.

```C#
#r "../QuantConnect.DataSource.QuiverQuantCongressTrading.dll"
using QuantConnect.DataSource;

var qb = new QuantBook();

// Add QuiverCongress
var aapl = qb.AddEquity("AAPL", Resolution.Daily).Symbol;
var quiverCongressSymbol = qb.AddData<QuiverCongress>(aapl).Symbol;

// Historical data
var history = qb.History<QuiverCongress>(quiverCongressSymbol, 60, Resolution.Daily);
foreach (var trade in history.SelectMany(x => x).OfType<QuiverCongressDataPoint>())
{
    Console.WriteLine($"{trade.EndTime} {trade.Symbol.Value} {trade.Representative} {trade.Amount}");
}

// Add Universe Selection
IEnumerable<Symbol> UniverseSelection(IEnumerable>BaseData> altCoarse)
{
    return from d in altCoarse.OfType<QuiverCongressDataPoint>()
        where d.Transaction == OrderDirection.Buy select d.Symbol;
}
var universe = qb.AddUniverse>QuiverQuantCongressUniverse>(UniverseSelection);

// Historical Universe data
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-60), qb.Time);
foreach (var trade in universeHistory.SelectMany(x => x).OfType<QuiverCongressDataPoint>())
{
    Console.WriteLine($"{trade.EndTime} {trade.Symbol.Value} {trade.Representative} {trade.Amount}");
}
```

Data Point Attributes

The US Congress Trading dataset provides `QuiverCongressDataPoint` , `QuiverCongress` , and `QuiverQuantCongressUniverse` objects.

QuiverCongressDataPoint Attributes

`QuiverCongressDataPoint` object has the following attributes:

QuiverCongress Attributes

`QuiverCongress` object has the following attributes:

QuiverQuantCongressUniverse Attributes

`QuiverQuantCongressUniverse` object has the following attributes:

Quiver Quantitative

US Government Contracts

Introduction

The US Government Contracts dataset by Quiver Quantitative tracks the transactions of government contracts with publicly traded companies. The data covers over 700 US Equities, starts in October 2019, and is delivered on a daily frequency. Quiver Quantitative creates this dataset by using the API for USASpending.gov , which has the official open data source of federal spending information. The rows in this dataset only show new contracts, not payments or modifications to existing contracts. The dollar amounts are based on the total dollars obligated from each contract.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the US Government Contracts dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Quiver Quantitative was founded by two college students in February 2020 with the goal of bridging the information gap between Wall Street and non-professional investors. Quiver allows retail investors to tap into the power of big data and have access to actionable, easy to interpret data that hasn't already been dissected by Wall Street.

Getting Started

The following snippet demonstrates how to request data from the US Government Contracts dataset:

```C#
_symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
_datasetSymbol = AddData<QuiverGovernmentContract>(_symbol).Symbol;

_universe = AddUniverse<QuiverGovernmentContractUniverse>(UniverseSelection);
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	October 15, 2019
Asset Coverage	748 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Requesting Data

To add US Government Contracts data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so

you can access the data later in your algorithm.

```
C#
public class QuiverGovernmentContractDataAlgorithm : QCAlgorithm
{
    private Symbol _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);
        var symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _datasetSymbol= AddData<QuiverGovernmentContract>(symbol).Symbol;
    }
}
```

Accessing Data

To get the current US Government Contract data, index the current `Slice` with the dataset `Symbol` . `Slice` objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var dataPoints = slice[_datasetSymbol];
        foreach (var dataPoint in dataPoints)
        {
            Log($"{_datasetSymbol} amount at {slice.Time}: {dataPoint.Amount}");
        }
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<QuiverGovernmentContract>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoints = kvp.Value;
        foreach(var dataPoint in dataPoints)
        {
            Log($"{datasetSymbol} amount at {slice.Time}: {dataPoint.Amount}");
        }
    }
}
```

Historical Data

To get historical US Government Contracts data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
var history = History<QuiverGovernmentContract>(_datasetSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on US Government Contract data, call the `AddUniverse` method with the `QuiverGovernmentContractUniverse` class and a selection function.

```C#
private Universe _universe;
public override void Initialize()
{
    _universe = AddUniverse<QuiverGovernmentContractUniverse>(altCoarse =>
    {
        var govContractDataBySymbol = new Dictionary<Symbol, List<QuiverGovernmentContractUniverse>>();

        foreach (var datum in altCoarse.OfType<QuiverGovernmentContractUniverse>())
        {
            var symbol = datum.Symbol;

            if (!govContractDataBySymbol.ContainsKey(symbol))
            {
                govContractDataBySymbol.Add(symbol, new List<QuiverGovernmentContractUniverse>());
            }
            govContractDataBySymbol[symbol].Add(datum);
        }

        return from kvp in govContractDataBySymbol
            where kvp.Value.Count >= 3 && kvp.Value.Sum(x => x.Amount) > 50000m
            select kvp.Key;
    });
}
```

Universe History

You can get historical universe data in an algorithm and in the Research Environment.

Historical Universe Data in Algorithms

To get historical universe data in an algorithm, call the `History` method with the `Universe` object and the lookback period. If there is no data in the period you request, the history result is empty.

```C#
var universeHistory = History(universe, 30, Resolution.Daily);
foreach (var contracts in universeHistory)
{
    foreach (QuiverGovernmentContractUniverse contract in contracts)
    {
        Log($"{contract.Symbol} amount at {contract.EndTime}: {contract.Amount}");
    }
}
```

Historical Universe Data in Research

To get historical universe data in research, call the `UniverseHistory` method with the `Universe` object, a start date, and an end date. This method returns the filtered universe. If there is no data in the period you request, the history result is empty.

```C#
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
foreach (var contracts in universeHistory)
{
    foreach (QuiverGovernmentContractUniverse contract in contracts)
    {
        Console.WriteLine($"{contract.Symbol} amount at {contract.EndTime}: {contract.Amount}");
    }
}
```

You can call the `History` method in Research.

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```
C#
RemoveSecurity(_datasetSymbol);
```

If you subscribe to US Government Contracts data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Quiver Quantitative US Government Contracts dataset enables you to create strategies using the latest information on government contracts activity. Examples include the following strategies:

- Buying securities that have received the most new government contract awards over the last month
- Trading securities that have had the biggest change in government contracts awards over the last year

Classic Algorithm Example

The following example algorithm buys Apple stock when they receive a new government contract worth over 50*K*. *Iftheyreceiveanewcontractworthunder*10K, the algorithm short sells Apple.

```
C#
public class QuiverGovernmentContractAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        _symbol = AddEquity("AAPL").Symbol;
        // Subscribe to government contract data for AAPL to generate trade signal
        _datasetSymbol = AddData<QuiverGovernmentContract>(_symbol).Symbol;

        // history request
        var history = History<QuiverGovernmentContract>(new[] {_datasetSymbol}, 10, Resolution.Daily);
        Debug($"We got {history.Count()} items from historical data request of {_datasetSymbol}.");
    }

    public override void OnData(Slice slice)
    {
        // Trade only base on government contract data
        foreach (var kvp in slice.Get<QuiverGovernmentContract>())
        {
            // Buy if over 50000 government contract amount, suggesting a large income
            if (kvp.Value.Any(x => (int) (x as QuiverGovernmentContract).Amount > 50000m))
            {
                SetHoldings(_symbol, 1);
            }
            // Sell if below 10000 government contract amount, suggesting a smaller than usual income
            else if (kvp.Value.Any(x => (int) (x as QuiverGovernmentContract).Amount < 10000m))
            {
                SetHoldings(_symbol, -1);
            }
        }
    }
}
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of US Equities that have just received a government contract worth at least $5K. Each day, it then forms an equal-weighted dollar-neutral portfolio with the 10 companies that received the largest contracts and the 10 companies that received the smallest contracts.

```
C#
```

```
public class QuiverGovernmentContractFrameworkAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Filter universe using government contract data
        AddUniverse<QuiverGovernmentContractUniverse>(data =>
        {
            var govContractDataBySymbol = new Dictionary<Symbol, List<QuiverGovernmentContractUniverse>>();

            foreach (var datum in data.OfType<QuiverGovernmentContractUniverse>())
            {
                var symbol = datum.Symbol;

                if (!govContractDataBySymbol.ContainsKey(symbol))
                {
                    govContractDataBySymbol.Add(symbol, new List<QuiverGovernmentContractUniverse>());
                }
                govContractDataBySymbol[symbol].Add(datum);
            }

            // Only select the stocks with over 5000 government contracts, which is considered material information
            return from kvp in govContractDataBySymbol
                where kvp.Value.Sum(x => x.Amount) > 5000m
                select kvp.Key;
        });

        // Custom alpha model that emit insights based on updated government contract data
        AddAlpha(new QuiverGovernmentContractAlphaModel());

        // Invest equally to evenly dissipate the capital concentration risk
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());

        SetExecution(new ImmediateExecutionModel());
    }
}

public class QuiverGovernmentContractAlphaModel: AlphaModel
{
    // A variable to control the rebalancing time
    private DateTime _time;
    // To hold the government contract dataset symbol for managing subscription
    private Dictionary<Symbol, Symbol> _datasetSymbolBySymbol = new();

    public QuiverGovernmentContractAlphaModel()
    {
        _time = DateTime.MinValue;
    }

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        if (_time > algorithm.Time) return new List<Insight>();

        // Trade signal only based on government contract data
        var dataPoints = slice.Get<QuiverGovernmentContract>();

        if (dataPoints.IsNullOrEmpty()) return new List<Insight>();

        // To aggregate all data per symbol for analysis
        var govContracts = dataPoints
            .Where(kvp => algorithm.Securities[kvp.Key.Underlying].Price != 0)
            .ToDictionary(kvp => kvp.Key, kvp => kvp.Value.Sum(x => ((QuiverGovernmentContract)x).Amount));

        // Long the top 10 highest government contract amount, predicting a higher expected income and return
        // Short the lowest 10 government contract amount, predicting a lower expected income and return
        var sortedByGovContract = from kvp in govContracts
                        orderby kvp.Value descending
                        select kvp.Key.Underlying;
        var longSymbols = sortedByGovContract.Take(10).ToList();
        var shortSymbols = sortedByGovContract.TakeLast(10).ToList();

        var insights = new List<Insight>();
        insights.AddRange(longSymbols.Select(symbol =>
            new Insight(symbol, Expiry.EndOfDay, InsightType.Price, InsightDirection.Up)));
        insights.AddRange(shortSymbols.Select(symbol =>
            new Insight(symbol, Expiry.EndOfDay, InsightType.Price, InsightDirection.Down)));
```

```
        _time = Expiry.EndOfDay(algorithm.Time);

        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            // Requesting government contract data for trade signal generation
            var symbol = security.Symbol;
            var datasetSymbol = algorithm.AddData<QuiverGovernmentContract>(symbol).Symbol;
            _datasetSymbolBySymbol.Add(symbol, datasetSymbol);
            // History request
            var history = algorithm.History<QuiverGovernmentContract>(datasetSymbol, 10, Resolution.Daily);
        }

        foreach (var security in changes.RemovedSecurities)
        {
            var symbol = security.Symbol;
            if (_datasetSymbolBySymbol.ContainsKey(symbol))
            {
                // Remove government contract data subscription to release computation resources
                _datasetSymbolBySymbol.Remove(symbol, out var datasetSymbol);
                algorithm.RemoveSecurity(datasetSymbol);
            }
        }
    }
}
```

Research Example

The following example lists all US Equities with Government contracts in the past year.

C#

```
#r "../QuantConnect.DataSource.QuiverGovernmentContracts.dll"
using QuantConnect.DataSource;

// Requesting data
var aapl = qb.AddEquity("AAPL", Resolution.Daily).Symbol;
var symbol = qb.AddData<QuiverGovernmentContract>(aapl).Symbol;

// Historical data
var history = qb.History<QuiverGovernmentContract>(symbol, 360, Resolution.Daily);
foreach (var contracts in history)
{
    foreach (QuiverGovernmentContract contract in contracts)
    {
        Console.WriteLine($"{contract.Symbol} amount at {contract.EndTime}: {contract.Amount}");
    }
}

// Add Universe Selection
IEnumerable<Symbol> UniverseSelection(IEnumerable<BaseData> altCoarse)
{
    return from d in altCoarse.OfType<QuiverGovernmentContractUniverse>()
        select d.Symbol;
}
var universe = qb.AddUniverse<QuiverGovernmentContractUniverse<(UniverseSelection);

// Historical Universe data
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-360), qb.Time);
foreach (var contracts in universeHistory)
{
    foreach (QuiverGovernmentContractUniverse contract in contracts)
    {
        Console.WriteLine($"{contract.Symbol} amount at {contract.EndTime}: {contract.Amount}");
    }
}
```

Data Point Attributes

The US Government Contracts dataset provides `QuiverGovernmentContract` and `QuiverGovernmentContractUniverse` objects.

QuiverGovernmentContract

`QuiverGovernmentContract` objects have the following attributes:

QuiverGovernmentContractUniverse

`QuiverGovernmentContractUniverse` objects have the following attributes:

Quiver Quantitative

WallStreetBets

Introduction

The WallStreetBets dataset by Quiver Quantitative tracks daily mentions of different equities on Reddit's popular WallStreetBets forum. The data covers 6,000 Equities, starts in August 2018, and is delivered on a daily frequency. The dataset is created by scraping the daily discussion threads on r/WallStreetBets and parsing the comments for ticker mentions.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the WallStreetBets dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Quiver Quantitative was founded by two college students in February 2020 with the goal of bridging the information gap between Wall Street and non-professional investors. Quiver allows retail investors to tap into the power of big data and have access to actionable, easy to interpret data that hasn't already been dissected by Wall Street.

Getting Started

The following snippet demonstrates how to request data from the WallStreetBets dataset:

```C#
_symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
_datasetSymbol = AddData<QuiverWallStreetBets>(_symbol).Symbol;

_universe = AddUniverse<QuiverWallStreetBetsUniverse>(UniverseSelection);
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	August 2018
Asset Coverage	6,000 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Requesting Data

To add WallStreetBets data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```
C#
public class QuiverWallStreetBetsDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);
        _symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _datasetSymbol = AddData<QuiverWallStreetBets>(_symbol).Symbol;
    }
}
```

Accessing Data

To get the current WallStreetBets data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var dataPoints = slice[_datasetSymbol];
        foreach (var dataPoint in dataPoints)
        {
            Log($"{_datasetSymbol} mentions at {slice.Time}: {dataPoint.Mentions}");
        }
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<QuiverWallStreetBets>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoints = kvp.Value;
        foreach (var dataPoint in dataPoints)
        {
            Log($"{datasetSymbol} mentions at {slice.Time}: {dataPoint.Mentions}");
        }
    }
}
```

Historical Data

To get historical WallStreetBets data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
var history = History<QuiverWallStreetBets>(_datasetSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on WallStreetBets data, call the `AddUniverse` method with the

`QuiverWallStreetBetsUniverse` class and a selection function.

```
C#

private Universe _universe;
public override void Initialize()
{
    _universe = AddUniverse<QuiverWallStreetBetsUniverse>(altCoarse =>
    {
        return from d in altCoarse.OfType<QuiverWallStreetBetsUniverse>()
            where d.Mentions > 10 && d.Rank > 10 select d.Symbol;
    });
}
```

For more information about dynamic universes, see Universes .

Universe History

You can get historical universe data in an algorithm and in the Research Environment.

Historical Universe Data in Algorithms

To get historical universe data in an algorithm, call the `History` method with the `Universe` object and the lookback period. If there is no data in the period you request, the history result is empty.

```
C#

var universeHistory = History(_universe, 30, Resolution.Daily);
foreach (var bets in universeHistory)
{
    foreach (QuiverWallStreetBetsUniverse bet in bets)
    {
        Log($"{bet.Symbol} mentions at {bet.EndTime}: {bet.Mentions}");
    }
}
```

Historical Universe Data in Research

To get historical universe data in research, call the `UniverseHistory` method with the `Universe` object, a start date, and an end date. This method returns the filtered universe. If there is no data in the period you request, the history result is empty.

```
C#

var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-30), qb.Time);
{
    foreach (QuiverWallStreetBetsUniverse bet in bets)
    {
        Log($"{bet.Symbol} rank at {bet.EndTime}: {bet.Rank}");
    }
}
```

You can call the `History` method in Research.

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```
C#

RemoveSecurity(_datasetSymbol);
```

If you subscribe to WallStreetBets data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The WallStreetBets dataset enables you to create strategies using the latest activity on the WallStreetBets daily discussion thread. Examples include the following strategies:

- Trading any security that is being mentioned
- Trading securities that are receiving more/less mentions than they were previously
- Trading the security that is being mentioned the most/least for the day

Classic Algorithm Example

The following example algorithm creates a dynamic universe of US Equities based on daily WallStreetBets data. When a security is mentioned on r/WallStreetBets more than five times in a day, the algorithm buys the security. When a security is mentioned five time in a day or less, the algorithm short sells the security.

```C#
public class QuiverWallStreetBetsDataAlgorithm : QCAlgorithm
{
    private Universe _universe;
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        UniverseSettings.Resolution = Resolution.Daily;
        // Filter using wall street bet insights
        _universe = AddUniverse<QuiverWallStreetBetsUniverse>(altCoarse =>
        {
            // Select the ones with popularity (mentions) of better-than-others performance (rank)
            return from d in altCoarse.OfType<QuiverWallStreetBetsUniverse>()
                    where d.Mentions > 10 && d.Rank < 100
                    select d.Symbol;
        });
    }

    public override void OnData(Slice slice)
    {
        var points = slice.Get<QuiverWallStreetBets>();
        foreach (var point in points.Values)
        {
            var symbol = point.Symbol.Underlying;
            var security = Securities[symbol];
            if (!security.IsTradable)
            {
                continue;
            }

            // Buy if the stock was mentioned more than 5 times in the WallStreetBets daily discussion, which
translate into high popularity of rise
            if (point.Mentions > 5 && !security.Holdings.IsLong)
            {
                MarketOrder(symbol, 1);
            }
            // Otherwise, short sell
            else if (point.Mentions <= 5 && !security.Holdings.IsShort)
            {
                MarketOrder(symbol, -1);
            }
        }
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach(var added in changes.AddedSecurities)
        {
            // Requesting wall street bet data to obtain the trader's insights
            var quiverWSBSymbol = AddData<QuiverWallStreetBets>(added.Symbol).Symbol;
            // Historical data
            var history = History<QuiverWallStreetBets>(quiverWSBSymbol, 60, Resolution.Daily);
        }
    }
}
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of US Equities based on daily WallStreetBets data. When a security is mentioned on r/WallStreetBets more than five times in a day, the algorithm buys the security. When a security is mentioned five time in a day or less, the algorithm short sells the security.

```C#
public class QuiverWallStreetBetsDataAlgorithm : QCAlgorithm
{
    private Universe _universe;
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
```

```
        UniverseSettings.Resolution = Resolution.Daily;
        // Filter using wall street bet insights
        _universe = AddUniverse<QuiverWallStreetBetsUniverse>(altCoarse =>
        {
            // Select the ones with popularity (mentions) of better-than-others performance (rank)
            return from d in altCoarse.OfType<QuiverWallStreetBetsUniverse>()
                    where d.Mentions > 10 && d.Rank < 100
                    select d.Symbol;
        });

        AddAlpha(new WallStreamBetsAlphaModel());

        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());

        AddRiskManagement(new NullRiskManagementModel());

        SetExecution(new ImmediateExecutionModel());
    }
}

public class WallStreamBetsAlphaModel : AlphaModel
{
    private Dictionary<Symbol, SymbolData> _symbolDataBySymbol = new Dictionary<Symbol, SymbolData>();
    private int _mentionsThreshold;

    public WallStreamBetsAlphaModel(int mentionsThreshold=5)
    {
        _mentionsThreshold = mentionsThreshold;
    }

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = new List<Insight>();

        var points = slice.Get<QuiverWallStreetBets>();
        foreach (var point in points.Values)
        {
            // Buy if the stock was mentioned more than 5 times in the WallStreetBets daily discussion, which
translate into high popularity of rise
            // Otherwise short sell
            var targetDirection = point.Mentions > _mentionsThreshold ? InsightDirection.Up : InsightDirection.Down;
            _symbolDataBySymbol[point.Symbol.Underlying].targetDirection = targetDirection;
        }

        foreach (var kvp in _symbolDataBySymbol)
        {
            var symbol = kvp.Key;
            var symbolData = kvp.Value;

            // Ensure we have security data for the current Slice to avoid stale fill
            if (!(slice.ContainsKey(symbol) && slice[symbol] != null))
            {
                continue;
            }

            if (symbolData.targetDirection != null)
            {
                insights.Add(Insight.Price(symbol, TimeSpan.FromDays(1),
(InsightDirection)symbolData.targetDirection));
                symbolData.targetDirection = null;
            }
        }
        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            var symbol = security.Symbol;
            _symbolDataBySymbol.Add(symbol, new SymbolData(algorithm, symbol));
        }

        foreach (var security in changes.RemovedSecurities)
        {
            var symbol = security.Symbol;
            if (_symbolDataBySymbol.ContainsKey(symbol))
            {
                _symbolDataBySymbol[symbol].dispose();
                 symbolDataBySymbol.Remove(symbol);
```

```
            }
        }
    }
}

public class SymbolData
{
    private Symbol _quiverWSBSymbol;
    private QCAlgorithm _algorithm;
    public InsightDirection? targetDirection = null;

    public SymbolData(QCAlgorithm algorithm, Symbol symbol)
    {
        _algorithm = algorithm;

        // Requesting wall street bet data to obtain the trader's insights
        _quiverWSBSymbol = algorithm.AddData<QuiverWallStreetBets>(symbol).Symbol;

        // Historical data
        var history = algorithm.History<QuiverWallStreetBets>(_quiverWSBSymbol, 60, Resolution.Daily);
    }

    public void dispose()
    {
        // Unsubscribe from the Quiver WallStreetBets feed for this security to release computationa resources
        _algorithm.RemoveSecurity(_quiverWSBSymbol);
    }
}
```

Research Example

The following example lists low-ranking US Equities that are mentioned more than ten times on r/WallStreetBets.

C#

```
#r "../QuantConnect.DataSource.QuiverWallStreetBets.dll"
using QuantConnect.DataSource;

var qb = new QuantBook();

// Requesting data
var aapl = qb.AddEquity("AAPL", Resolution.Daily).Symbol;
var symbol = qb.AddData<QuiverWallStreetBets>(aapl).Symbol;

// Historical data
var history = qb.History<QuiverWallStreetBets>(symbol, 60, Resolution.Daily);
foreach (var bet in history.OfType<QuiverWallStreetBets>())
{
    Console.WriteLine($"{bet.Symbol} rank at {bet.EndTime}: {bet.Rank}");
}

// Add Universe Selection
IEnumerable<Symbol> UniverseSelection(IEnumerable<BaseData> altCoarse)
{
    return from d in altCoarse.OfType<QuiverWallStreetBetsUniverse>()
        where d.Mentions > 10 && d.Rank < 100 select d.Symbol;
}
var universe = qb.AddUniverse<QuiverWallStreetBetsUniverse>(UniverseSelection);

// Historical Universe data
var universeHistory = qb.UniverseHistory(universe, qb.Time.AddDays(-60), qb.Time);
foreach (var bets in universeHistory)
{
    foreach (QuiverWallStreetBetsUniverse bet in bets)
    {
        Console.WriteLine($"{bet.Symbol} rank at {bet.EndTime}: {bet.Rank}");
    }
}
```

Data Point Attributes

The WallStreetBets dataset provides `QuiverWallStreetBets` and `QuiverWallStreetBetsUniverse` objects.

QuiverWallStreetBets Attributes

`QuiverWallStreetBets` objects have the following attributes:

QuiverWallStreetBetsUniverse Attributes

`QuiverWallStreetBetsUniverse` objects have the following attributes:

Datasets

RegAlytics

RegAlytics was founded by Mary Kopczynski, Aaron Heisler, Alexander Appugliese, and Werner Pauliks in 2019 with the goal of significantly reducing the time and cost required to mitigate regulatory risk. RegAlytics provides access to accurate and clean regulatory data from all global regulators in all sectors that is enriched by regulatory experts for risk and compliance teams everywhere. Please come to RegAlytics directly if you would like data on other sectors or countries!

US Regulatory Alerts - Financial Sector

RegAlytics

US Regulatory Alerts - Financial Sector

Introduction

The US Regulatory Alerts dataset by RegAlytics tracks changes from over 8,000 globally governing bodies. The data covers over 2.5 million alerts, starts from January 2020, and is delivered on a daily basis. This dataset is created by sourcing information from over 8,000 regulators and using proprietary technology to gather and structure the regulatory data. Once prepared, the data is thoroughly reviewed by RegAlytics' team of regulatory experts and delivered each morning by 8AM for industry use.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the US Regulatory Alerts - Financial Sector dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

RegAlytics was founded by Mary Kopczynski, Aaron Heisler, Alexander Appugliese, and Werner Pauliks in 2019 with the goal of significantly reducing the time and cost required to mitigate regulatory risk. RegAlytics provides access to accurate and clean regulatory data from all global regulators in all sectors that is enriched by regulatory experts for risk and compliance teams everywhere. Please come to RegAlytics directly if you would like data on other sectors or countries!

Getting Started

The following snippet demonstrates how to request data from the US Regulatory Alerts - Financial Sector dataset:

```C#
_datasetSymbol = AddData<RegalyticsRegulatoryArticles>("REG").Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2020
Coverage	2,500,000 Alerts
Data Density	Sparse
Resolution	Daily
Timezone	New York

Requesting Data

To add US Regulatory Alerts data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can

access the data later in your algorithm.

```
C#
public class RegalyticsDataAlgorithm : QCAlgorithm
{
    private Symbol _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2021, 1, 1);
        SetEndDate(2021, 6, 1);
        SetCash(100000);

        _datasetSymbol = AddData<RegalyticsRegulatoryArticles>("REG").Symbol;
    }
}
```

Accessing Data

To get the current US Regulatory Alerts data, index the current `Slice` with the dataset `Symbol` . `Slice` objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var dataPoints = slice[_datasetSymbol];
        foreach (var dataPoint in dataPoints)
        {
            Log($"{_datasetSymbol} title at {slice.Time}: {dataPoint.Title}");
        }
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    var data = slice.Get<RegalyticsRegulatoryArticles>();
    if (!data.IsNullOrEmpty())
    {
        foreach (var articles in data.Values)
        {
            Log($"{Time} {articles.ToString()}");
            foreach (RegalyticsRegulatoryArticle article in articles)
            {
                Log($"{_datasetSymbol} article title at {slice.Time}: {article.Title}");
            }
        }
    }
}
```

Historical Data

To get historical US Regulatory Alerts data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
var history = History<RegalyticsRegulatoryArticles>(_datasetSymbol, 100, Resolution.Daily);
```

Remove Subscriptions

To remove your subscription to US Regulatory Alerts data, call the `RemoveSecurity` method.

```
RemoveSecurity(_datasetSymbol);
```

Example Applications

This regulatory dataset enables you to accurately design strategies while mitigating regulatory risk. Examples include the following strategies:

- Temporarily increasing/decreasing exposure to securities when new regulations are announced
- Parsing the content of regulatory announcements to determine market or sector impact
- Country rotation based on regulatory alert sentiment

Classic Algorithm Example

The following example algorithm sells short 100% SPY if any predetermined negative sentimental phrase was present in any of the previous day's regulatory articles. Otherwise, it buys and hold 100% SPY.

```
public class RegalyticsDataAlgorithm : QCAlgorithm
{
    // Pre-defined list of negative sentiment phrases as indicator for short selling, since these words will decrease
confident in equity performance
    private readonly string[] _negativeSentimentPhrases = new [] {"emergency rule", "proposed rule change",
"development of rulemaking"};
    private Symbol _symbol, _regalyticsSymbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        _symbol = AddEquity("SPY", extendedMarketHours: true).Symbol;

        // Requesting data to receive updated regulatory news for timely short selling
        _regalyticsSymbol = AddData<RegalyticsRegulatoryArticles>("REG").Symbol;

        // Historical data for past articles
        var history = History<RegalyticsRegulatoryArticles>(_regalyticsSymbol, 7, Resolution.Daily);
        Debug($"We got {history.Count()} items from our history request");
    }

    public override void OnData(Slice slice)
    {
        // Only trade on regulatory news data
        var data = slice.Get<RegalyticsRegulatoryArticles>();
        if (data.IsNullOrEmpty())
        {
            return;
        }
        // If any of the negative phrases appeared in regulatory news, we expect a market drop for the day
        var detectedNegativity = false;
        foreach (var articles in data.Values)
        {
            if (detectedNegativity) break;
            detectedNegativity = _negativeSentimentPhrases
                .Any(p => articles.Any(x => ((RegalyticsRegulatoryArticle) x).Title.ToLower().Contains(p)));
        }
        Transactions.CancelOpenOrders();
        var quantity = CalculateOrderQuantity(_symbol, detectedNegativity ? -1d : 1d);
        if (quantity == 0) return;
        MarketOrder(_symbol, quantity);
    }
}
```

Framework Algorithm Example

The following example algorithm buys and holds the S&P 500 ETF, SPY. To reduce regulatory risk, it liquidates your holdings for 2 days following regulatory announcements regarding new rules.

C#

```
public class RegalyticsDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        UniverseSettings.Resolution = Resolution.Daily;
        //  Trade SPY as the proxy of the whole equity market movement on the regulatory news
        AddUniverseSelection(
            new ManualUniverseSelectionModel(
                QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA)
        ));

        AddAlpha(new RegAnalyticsAlphaModel(this));

        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel(time => null));

        AddRiskManagement(new NullRiskManagementModel());

        SetExecution(new ImmediateExecutionModel());
    }

    public class RegAnalyticsAlphaModel : AlphaModel
    {
        private List<Symbol> _symbols = new List<Symbol>();
        private Symbol _regalyticsSymbol;
        private DateTime _lastNewsDate = DateTime.MinValue;
        private InsightDirection _targetDirection = InsightDirection.Up;
        // Pre-defined list of negative sentiment phrases as indicator for short selling, since these words will
decrease confident in equity performance
        private string[] _negativeSentimentPhrases = new string[] {"emergency rule", "proposed rule change",
"development of rulemaking"};
        // Assuming the negative impact of the regulatory news will last for 2 days
        private TimeSpan _newsAffectDuration = TimeSpan.FromDays(2);

        public RegAnalyticsAlphaModel(QCAlgorithm algorithm)
        {
            // Requesting data to receive updated regulatory news for timely short selling
            _regalyticsSymbol = algorithm.AddData<RegalyticsRegulatoryArticles>("REG").Symbol;

            // Historical data for past articles
            var history = algorithm.History<RegalyticsRegulatoryArticles>(_regalyticsSymbol, 14, Resolution.Daily);
            algorithm.Debug($"We got {history.Count()} items from our history request");
        }

        public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
        {
            var insights = new List<Insight>();

            // Only trade on regulatory news data
            if (slice.ContainsKey(_regalyticsSymbol) && slice[_regalyticsSymbol] != null)
            {
                var articles = slice[_regalyticsSymbol];
                foreach (RegalyticsRegulatoryArticle article in articles)
                {
                    var title = article.Title.ToLower();
                    // If any of the negative phrases appeared in regulatory news, we expect a market drop for the
day
                    // Signal an exit from the market when regulatory articles with negative sentiment are released
                    if (_negativeSentimentPhrases.Any(x => title.Contains(x)))
                    {
                        _targetDirection = InsightDirection.Flat;
                        _lastNewsDate = slice.Time;
                    }
                }
            }

            // Signal an entry if we've waited long enough for the market to digest the negative news
            if (_lastNewsDate + _newsAffectDuration < slice.Time)
            {
                _targetDirection = InsightDirection.Up;
```

```
                _targetDirection = InsightDirection.Up;
            }

            foreach (var symbol in _symbols)
            {
                // Ensure we have security data in the current Slice to avoid stale fill
                if (!(slice.ContainsKey(symbol) && slice[symbol] != null))
                {
                    continue;
                }

                if (_targetDirection == InsightDirection.Up && !algorithm.Portfolio[symbol].Invested)
                {
                    insights.Add(Insight.Price(symbol, TimeSpan.FromDays(90), InsightDirection.Up));
                }
                else if (_targetDirection == InsightDirection.Flat && algorithm.Portfolio[symbol].Invested)
                {
                    insights.Add(Insight.Price(symbol, _newsAffectDuration, InsightDirection.Flat));
                }
            }
            return insights;
        }

        public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
        {
            foreach (var security in changes.AddedSecurities)
            {
                _symbols.Add(security.Symbol);
            }

            foreach (var security in changes.RemovedSecurities)
            {
                _symbols.Remove(security.Symbol);
            }
        }
    }
}
```

Data Point Attributes

The US Regulatory Alerts dataset provides `RegalyticsRegulatoryArticle` and `RegalyticsRegulatoryArticles` objects.

RegalyticsRegulatoryArticle

`RegalyticsRegulatoryArticle` objects have the following attributes:

RegalyticsRegulatoryArticles

`RegalyticsRegulatoryArticles` objects have the following attributes:

Datasets

Securities and Exchange Commission

The mission of the U.S. Securities and Exchange Commission is to protect investors, maintain fair, orderly, and efficient markets, and facilitate capital formation. The SEC oversees the key participants in the securities world, including securities exchanges, securities brokers and dealers, investment advisors, and mutual funds. The SEC is concerned primarily with promoting the disclosure of important market-related information, maintaining fair dealing, and protecting against fraud.

US SEC Filings

Securities and Exchange Commission

US SEC Filings

Introduction

The US SEC Filings dataset provides the quarterly financial earning reports that the United States Securities and Exchange Commission (SEC) requires from publicly traded companies in the US. The data covers 15,000 US Equities, starts in January 1998, and is delivered on a daily frequency. The data is sourced from the SEC's Electronic Data Gathering, Analysis, and Retrieval (EDGAR) system. QuantConnect downloads and formats the Quarterly Financial Reports (10-Q) and Annual Financial Report (8-K) filings of companies into a format for easy consumption by LEAN.

For more information about the US SEC Filings dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

The mission of the U.S. Securities and Exchange Commission is to protect investors, maintain fair, orderly, and efficient markets, and facilitate capital formation. The SEC oversees the key participants in the securities world, including securities exchanges, securities brokers and dealers, investment advisors, and mutual funds. The SEC is concerned primarily with promoting the disclosure of important market-related information, maintaining fair dealing, and protecting against fraud.

Getting Started

The following snippet demonstrates how to request data from the US SEC Filings dataset:

```C#
_symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
_report8KSymbol = AddData<SECReport8K>(_symbol, Resolution.Daily).Symbol;
_report10KSymbol = AddData<SECReport10K>(_symbol, Resolution.Daily).Symbol;
_report10QSymbol = AddData<SECReport10Q>(_symbol, Resolution.Daily).Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 1998
Asset Coverage	15,000 US Equities
Data Density	Sparse
Resolution	Daily
Timezone	UTC

Requesting Data

To add US SEC Filings data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```
C#
public class SECReportAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _report8KSymbol, _report10KSymbol, _report10QSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2019, 8, 21);
        SetCash(100000);

        _symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        _report8KSymbol = AddData<SECReport8K>(_symbol, Resolution.Daily).Symbol;
        _report10KSymbol = AddData<SECReport10K>(_symbol, Resolution.Daily).Symbol;
        _report10QSymbol = AddData<SECReport10Q>(_symbol, Resolution.Daily).Symbol;
    }
}
```

Accessing Data

To get the current US SEC Filings data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_report8KSymbol))
    {
        var dataPoint = slice[_report8KSymbol];
        Log($"{_report8KSymbol} report count at {slice.Time}: {data_point.Report.Documents.Count}");
    }

    if (slice.ContainsKey(_report10KSymbol))
    {
        var dataPoint = slice[_report10KSymbol];
        Log($"{_report10KSymbol} report count at {slice.Time}: {data_point.Report.Documents.Count}");
    }

    if (slice.ContainsKey(_report10QSymbol))
    {
        var dataPoint = slice[_report10QSymbol];
        Log($"{_report10QSymbol} report count at {slice.Time}: {data_point.Report.Documents.Count}");
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<SECReport8K>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} report count at {slice.Time}: {dataPoint.Report.Documents.Count}");
    }

    foreach (var kvp in slice.Get<SECReport10K>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} report count at {slice.Time}: {dataPoint.Report.Documents.Count}");
    }

    foreach (var kvp in slice.Get<SECReport10Q>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} report count at {slice.Time}: {dataPoint.Report.Documents.Count}");
    }
}
```

Historical Data

To get historical US SEC Filings data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
// Dataset objects
var report8KHistory = History<SECReport8K>(_report8KSymbol, 100, Resolution.Daily);
var report10KHistory = History<SECReport10K>(_report10KSymbol, 100, Resolution.Daily);
var report10QHistory = History<SECReport10Q>(_report10QSymbol, 100, Resolution.Daily);

// Slice objects
var history = History(new[] {_report8KSymbol, _report10KSymbol, _report10QSymbol}, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_report8KSymbol);
RemoveSecurity(_report10KSymbol);
RemoveSecurity(_report10QSymbol);
```

If you subscribe to US SEC Filings data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The US SEC Filings dataset enables you to create strategies using information from SEC reports. Examples include the following strategies:

- Extracting information about corporate earnings from the documents for further analysis
- Applying sentiment analysis to the text content of the documents (for example, keyword scoring and ranking)

Classic Algorithm Example

The following example algorithm creates a dynamic universe of the 10 most liquid US Equities. It then buys the securities that have an 8K report with more than 20,000 characters and forms an equal-weighted portfolio. Instead of trading based on how long the report is, you could use sentiment analysis by scoring keywords.

```C#
public class SECReport8KAlgorithm : QCAlgorithm
{
    private Dictionary<Symbol, Symbol> _datasetSymbolBySymbol = new Dictionary<Symbol, Symbol>();
    private List<Symbol> _longSymbols = new List<Symbol>();
    private bool _rebalance = false;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        UniverseSettings.Resolution = Resolution.Daily;
        AddUniverse(CoarseSelector);

        // Request underlying equity data.
        var ibm = AddEquity("IBM", Resolution.Daily).Symbol;
        // Add SEC report 10-Q data for the underlying IBM asset for trade signal generation
        var earningsFiling = AddData<SECReport10Q>(ibm, Resolution.Daily).Symbol;
        // Request 120 days of history with the SECReport10Q IBM custom data Symbol.
        var history = History<SECReport10Q>(earningsFiling, 120, Resolution.Daily);
        Debug($"We got {history.Count()} items from our history request");
    }

    public IEnumerable<Symbol> CoarseSelector(IEnumerable<CoarseFundamental> coarse)
    {
        // Select the top 10 traded companies with fundamental data, since the market activities can better reflect
the fundamental factor efficiently
        return coarse.Where(x => x.HasFundamentalData)
            .OrderByDescending(x => x.DollarVolume)
            .Take(10).Select(x => x.Symbol);
    }

    public override void OnData(Slice slice)
    {
        // Trade from SEC data
        foreach (var report in slice.Get<SECReport8K>().Values)
        {
            var underlyingSymbol = report.Symbol.Underlying;
            // Skip the Symbol if it's no longer in the universe (insuffucient popularity to reach market efficiency
of fundamental factor)
            if (!_datasetSymbolBySymbol.ContainsKey(underlyingSymbol))
            {
                if (_longSymbols.Contains(underlyingSymbol))
                {
                    _rebalance = true;
                    _longSymbols.Remove(underlyingSymbol);
                }
                continue;
            }

            // Long the stocks with SEC report over 20000 text length, suggesting excessive insights or future
projections are highlighted
            var reportTextLength = report.Report.Documents.Select(x => x.Text.Length).Sum();
            if (reportTextLength > 20000)
            {
                if (!_longSymbols.Contains(underlyingSymbol))
                {
                    _rebalance = true;
                    _longSymbols.Add(underlyingSymbol);
                }
            }
            else if (_longSymbols.Contains(underlyingSymbol))
            {
                _rebalance = true;
                _longSymbols.Remove(underlyingSymbol);
            }
        }

        if (!_rebalance)
        {
            return;
```

```
        }
        _rebalance = false;

        // Invest equally to evenly dissipate the capital concentration risk
        var portfolioTargets = new List<PortfolioTarget>();
        var equalWeighting = _longSymbols.Count > 0 ? 1.0m / _longSymbols.Count : 0m;
        foreach (var kvp in Portfolio)
        {
            var symbol = kvp.Key;
            var securityHolding = kvp.Value;
            var weight = 0m;
            if (_longSymbols.Contains(symbol))
            {
                weight = equalWeighting;
            }
            else if (!securityHolding.Invested)
            {
                continue;
            }
            portfolioTargets.Add(new PortfolioTarget(symbol, weight));
        }
        SetHoldings(portfolioTargets);
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            // If added to universe, adds SECReport8K for trade signal generation
            _datasetSymbolBySymbol[security.Symbol] = AddData<SECReport8K>(security.Symbol).Symbol;
        }

        foreach (var security in changes.RemovedSecurities)
        {
            Symbol reportSymbol;
            if (_datasetSymbolBySymbol.TryGetValue(security.Symbol, out reportSymbol))
            {
                // Remove SEC data subscription to release computation resources
                RemoveSecurity(reportSymbol);
                _datasetSymbolBySymbol.Remove(security.Symbol);
            }
        }
    }
}
```

Framework Algorithm Example

The following example algorithm creates a dynamic universe of the 10 most liquid US Equities. It then buys the securities that have an 8K report with more than 20,000 characters and forms an equal-weighted portfolio. Instead of trading based on how long the report is, you could use sentiment analysis by scoring keywords.

C#

```
public class SECReport8KAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        UniverseSettings.Resolution = Resolution.Daily;
        // Filter for the most liquid stocks for trading due to high market activities for efficient pricing from
fundamental factor
        AddUniverseSelection(new MostLiquidFundamentalUniverseSelectionModel());
        // Custom alpha model that generates insights from SEC data
        AddAlpha(new SECReport8KAlphaModel());
        // Invest equally to evenly dissipate the capital concentration risk
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel(time => null));
    }
}

class MostLiquidFundamentalUniverseSelectionModel : FundamentalUniverseSelectionModel
{
    public override IEnumerable<Symbol> Select(QCAlgorithm algorithm, IEnumerable<Fundamental> fundamental)
    {
```

```
        {
            // Select the top 10 traded companies with fundamental data, since the market activities can better reflect
the fundamental factor efficiently
            return fundamental
                .OrderByDescending(f => f.DollarVolume)
                .Take(10)
                .Select(f => f.Symbol);
        }
    }

    public class SECReport8KAlphaModel : AlphaModel
    {
        private readonly Dictionary<Symbol, Symbol> _datasetSymbolBySymbol = [];

        public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
        {
            var insights = new List<Insight>();

            // Trade only based on SEC data
            foreach (var report in slice.Get<SECReport8K>().Values)
            {
                var underlyingSymbol = report.Symbol.Underlying;
                if (!_datasetSymbolBySymbol.ContainsKey(underlyingSymbol))
                {
                    continue;
                }

                // Long the stocks with SEC report over 20000 text length, suggesting excessive insights or future
projections are highlighted
                var reportTextLength = report.Report.Documents.Select(x => x.Text.Length).Sum();
                InsightDirection direction;
                if (reportTextLength > 20000)
                {
                    direction = InsightDirection.Up;
                }
                else if (algorithm.Portfolio[underlyingSymbol].Invested)
                {
                    direction = InsightDirection.Flat;
                }
                else
                {
                    continue;
                }
                insights.Add(Insight.Price(underlyingSymbol, TimeSpan.FromDays(30), direction));
            }

            return insights;
        }

        public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
        {
            foreach (var symbol in changes.AddedSecurities.Select(x => x.Symbol))
            {
                // If added to universe, adds SECReport8K for trade signal generation
                _datasetSymbolBySymbol[symbol] = algorithm.AddData<SECReport8K>(symbol).Symbol;
            }

            foreach (var symbol in changes.RemovedSecurities.Select(x => x.Symbol))
            {
                // Remove SEC data subscription to release computation resources
                if (_datasetSymbolBySymbol.TryGetValue(symbol, out Symbol reportSymbol))
                {
                    algorithm.RemoveSecurity(reportSymbol);
                    _datasetSymbolBySymbol.Remove(symbol);
                }
            }
        }
    }
```

Data Point Attributes

The US SEC Filings dataset provides `SECReport8K` , `SECReport10K` , and `SECReport10Q` objects.

Report 8K Attributes

`SECReport8K` objects have the following attributes:

Report 10K Attributes

`SECReport10K` objects have the following attributes:

Report 10Q Attributes

`SECReport10Q` objects have the following attributes:

Datasets

Smart Insider

Smart Insider was founded by Michael Tindale in 2016 with the goal of forming the most progressive insider data vendor in the field. Smart Insider provides access to buyback intention and transactions for quantitative researchers. In addition to their Corporate Buybacks dataset, Smart Insider provides data on stock trades made by US politicians and thousands of high net worth individuals around the globe.

Corporate Buybacks

Smart Insider

Corporate Buybacks

Introduction

The Corporate Buybacks dataset by Smart Insider tracks US Equities share buyback programs. The data covers 3,000 US Equities, starts in May 2015, and is delivered on a second frequency. This dataset is created by analyzing daily buyback announcements and by using secondary data sources to ensure records are accurate and complete.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Corporate Buybacks dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Smart Insider was founded by Michael Tindale in 2016 with the goal of forming the most progressive insider data vendor in the field. Smart Insider provides access to buyback intention and transactions for quantitative researchers. In addition to their Corporate Buybacks dataset, Smart Insider provides data on stock trades made by US politicians and thousands of high net worth individuals around the globe.

Getting Started

The following snippet demonstrates how to request data from the Corporate Buybacks dataset:

```C#
_symbol = AddEquity("AAPL", Resolution.Minute).Symbol;
_intentionSymbol = AddData<SmartInsiderIntention>(_symbol).Symbol;
_transactionSymbol = AddData<SmartInsiderTransaction>(_symbol).Symbol;

_intentionUniverse = AddUniverse<SmartInsiderIntentionUniverse>(IntentionSelection);
_transactionUniverse = AddUniverse<SmartInsiderTransactionUniverse>(TransactionSelection);
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	May 2015
Asset Coverage	3,000 US Equities
Data Density	Sparse
Resolution	Second
Timezone	New York

Requesting Data

To add Corporate Buybacks data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can

access the data later in your algorithm.

```
C#
public class CorporateBuybacksDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _intentionSymbol, _transactionSymbol;

    public override void Initialize()
    {
        SetStartDate(2016, 1, 1);
        SetEndDate(2021, 1, 1);
        SetCash(100000);

        _symbol = AddEquity("AAPL", Resolution.Minute).Symbol;
        _intentionSymbol = AddData<SmartInsiderIntention>(_symbol).Symbol;
        _transactionSymbol = AddData<SmartInsiderTransaction>(_symbol).Symbol;
    }
}
```

Accessing Data

To get the current Corporate Buybacks data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_intentionSymbol))
    {
        var dataPoint = slice[_intentionSymbol];
        Log($"{_intentionSymbol} intention amount at {slice.Time}: {dataPoint.Amount}");
    }

    if (slice.ContainsKey(_transactionSymbol))
    {
        var dataPoint = slice[_transactionSymbol];
        Log($"{_transactionSymbol} transaction amount at {slice.Time}: {dataPoint.Amount}");
    }
}
```

To iterate through all of the dataset objects in the current `Slice` , call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<SmartInsiderIntention>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} intention amount at {slice.Time}: {dataPoint.Amount}");
    }

    foreach (var kvp in slice.Get<SmartInsiderTransaction>())
    {
        var datasetSymbol = kvp.Key;
        var dataPoint = kvp.Value;
        Log($"{datasetSymbol} transaction amount at {slice.Time}: {dataPoint.Amount}");
    }
}
```

Historical Data

To get historical Corporate Buybacks data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
// Dataset objects
var intentionHistory = History<SmartInsiderIntention>(_intentionSymbol, 100, Resolution.Daily);
var transactionHistory = History<SmartInsiderTransaction>(_transactionSymbol, 100, Resolution.Daily);

// Slice objects
var history = History(new[] {_intentionSymbol, _transactionSymbol}, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Universe Selection

To select a dynamic universe of US Equities based on Corporate Buybacks data, call the `AddUniverse` method with the `SmartInsiderIntentionUniverse` class or the `SmartInsiderTransactionUniverse` and a selection function.

```
private Universe _intentionUniverse;
private Universe _transactionUniverse;
public override void Initialize()
{
    _intentionUniverse = AddUniverse<SmartInsiderIntentionUniverse>(altCoarse =>
    {
        return from d in altCoarse.OfType<SmartInsiderIntentionUniverse>()
            where d.Percentage > 0.005m && d.USDMarketCap > 100000000m
            select d.Symbol;
    });
    _transactionUniverse = AddUniverse<SmartInsiderTransactionUniverse>(altCoarse =>
    {
        return from d in altCoarse.OfType<SmartInsiderTransactionUniverse>()
            where d.BuybackPercentage > 0.005m && d.USDMarketCap > 100000000m
            select d.Symbol;
    });
}
```

For more information about dynamic universes, see Universes .

Universe History

You can get historical universe data in an algorithm and in the Research Environment.

Historical Universe Data in Algorithms

To get historical universe data in an algorithm, call the `History` method with the `Universe` object and the lookback period. If there is no data in the period you request, the history result is empty.

```
var intentionUniverseHistory = History(_intentionUniverse, 30, Resolution.Daily);
foreach (var intentions in intentionUniverseHistory)
{
    foreach (SmartInsiderIntentionUniverse intention in intentions)
    {
        Log($"{intention.Symbol.Value} amount at {intention.EndTime}: {intention.AmountValue}");
    }
}

var transactionUniverseHistory = History(_transactionUniverse, 30, Resolution.Daily);
foreach (var transactions in transactionHistory)
{
    foreach (SmartInsiderTransactionUniverse transaction in transactions)
    {
        Log($"{transaction.Symbol.Value} amount at {transaction.EndTime}: {transaction.Amount}");
    }
}
```

Historical Universe Data in Research

To get historical universe data in research, call the `UniverseHistory` method with the `Universe` object, a start date, and an end date. This method returns the filtered universe. If there is no data in the period you request, the history result is empty.

```C#
var intentionUniverseHistory = qb.UniverseHistory(intentionUniverse, qb.Time.AddDays(-30), qb.Time);
foreach (var intentions in intentionUniverseHistory)
{
    foreach (SmartInsiderIntentionUniverse intention in intentions)
    {
        Console.WriteLine($"{intention.Symbol.Value} amount at {intention.EndTime}: {intention.AmountValue}");
    }
}

var transactionUniverseHistory = qb.UniverseHistory(transactionUniverse, qb.Time.AddDays(-30), qb.Time);
foreach (var transactions in transactionHistory)
{
    foreach (SmartInsiderTransactionUniverse transaction in transactions)
    {
        Console.WriteLine($"{transaction.Symbol.Value} amount at {transaction.EndTime}: {transaction.Amount}");
    }
}
```

You can call the `History` method in Research.

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_intentionSymbol);
RemoveSecurity(_transactionSymbol);
```

If you subscribe to Corporate Buybacks data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Corporate Buybacks dataset enables you to design strategies using information on company buyback programs. Examples include the following strategies:

- Buying securities when the company announces an upcoming share buyback on the premise that the reduction in supply (shares outstanding) will drive up the remaining shares' price
- Buying securities when the company executes an upcoming share buyback on the premise that the reduction in supply (shares outstanding) will drive up the remaining shares' price

Classic Algorithm Example

The following example algorithm buys Apple stock when there is a buyback intention or transaction and then it sells the stock 3 days later:

```C#
public class CorporateBuybacksDataAlgorithm : QCAlgorithm
{
    private Symbol _aapl;
    private Symbol _smartInsiderIntention;
    private Symbol _smartInsiderTransaction;
    private DateTime _entryTime;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        _aapl = AddEquity("AAPL", Resolution.Minute).Symbol;

        // Requesting insider trade intention news and actual trades to estimate the return, since insiders may have
better information of the future confidence
        _smartInsiderIntention = AddData<SmartInsiderIntention>(_aapl).Symbol;
        _smartInsiderTransaction = AddData<SmartInsiderTransaction>(_aapl).Symbol;

        // Historical data
        var intentionHistory = History<SmartInsiderIntention>(_smartInsiderIntention, 365, Resolution.Daily);
        Debug($"We got {intentionHistory.Count()} items from our history request for intentions");

        var transactionHistory = History<SmartInsiderTransaction>(_smartInsiderTransaction, 365, Resolution.Daily);
        Debug($"We got {transactionHistory.Count()} items from our history request for transactions");
    }

    public override void OnData(Slice slice)
    {
        // Buy Apple whenever we receive a buyback intention or transaction notification, given the insiders may have
confidence in the future to buy more
        // This news may stimulate market popularity
        if (slice.ContainsKey(_smartInsiderIntention) || slice.ContainsKey(_smartInsiderTransaction))
        {
            SetHoldings(_aapl, 1);
            _entryTime = Time;
        }

        // Liquidate holdings 3 days after the latest entry
        // The market popularity and possible overbrought is cooled
        if (Portfolio.Invested && Time >= _entryTime + TimeSpan.FromDays(3))
        {
            Liquidate();
        }
    }
}
```

Framework Algorithm Example

The following example algorithm buys Apple stock when there is a buyback intention or transaction and then it sells the stock 3 days later:

```C#
public class CorporateBuybacksDataAlgorithm : QCAlgorithm
{

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Only interest in AAPL's insider trades information
        var symbols = new[] {QuantConnect.Symbol.Create("AAPL", SecurityType.Equity, Market.USA)};
        AddUniverseSelection(new ManualUniverseSelectionModel(symbols));
        AddAlpha(new CorporateBuybacksAlphaModel());
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }

    public class CorporateBuybacksAlphaModel : AlphaModel
    {
        private Symbol _aapl;
        private Symbol _smartInsiderIntention;
        private Symbol _smartInsiderTransaction;

        public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
        {
            var insights = new List<Insight>();

            // Buy Apple whenever we receive a buyback intention or transaction notification, given the insiders may
have confidence in the future to buy more
            // This news may stimulate market popularity, so we hold for 3 days
            if (slice.ContainsKey(_smartInsiderIntention) || slice.ContainsKey(_smartInsiderTransaction))
            {
                insights.Add(Insight.Price(_aapl, TimeSpan.FromDays(3),  InsightDirection.Up));
            }
            return insights;
        }

        public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
        {
            foreach (var security in changes.AddedSecurities)
            {
                _aapl = security.Symbol;

                // Requesting insider trade intention news and actual trades to estimate the return, since insiders
may have better information of the future confidence
                _smartInsiderIntention = algorithm.AddData<SmartInsiderIntention>(_aapl).Symbol;
                _smartInsiderTransaction = algorithm.AddData<SmartInsiderTransaction>(_aapl).Symbol;

                // Historical data
                var intentionHistory = algorithm.History<SmartInsiderIntention>(_smartInsiderIntention, 365,
Resolution.Daily);
                algorithm.Debug($"We got {intentionHistory.Count()} items from our history request for intentions");

                var transactionHistory = algorithm.History<SmartInsiderTransaction>(_smartInsiderTransaction, 365,
Resolution.Daily);
                algorithm.Debug($"We got {transactionHistory.Count()} items from our history request for
transactions");
            }
        }
    }
}
```

Research Example

The following example lists US Equities with the highest buyback amount.

C#

```
#r "../QuantConnect.DataSource.SmartInsiderIntentionsTransactions.dll"
using QuantConnect.DataSource;

var qb = new QuantBook();

// Requesting data
var symbol = qb.AddEquity("AAPL").Symbol;
var intentionSymbol = qb.AddData<SmartInsiderIntention>(symbol).Symbol;
var transactionSymbol = qb.AddData<SmartInsiderTransaction>(symbol).Symbol;

// Historical data for SmartInsiderIntention
var intentionHistory = qb.History<SmartInsiderIntention>(symbol, 300, Resolution.Daily);
foreach (SmartInsiderIntention intention in intentionHistory)
{
    Console.WriteLine($"{intention.NoteText} at {intention.EndTime}");
}

// Historical data for SmartInsiderTransaction
var transactionHistory = qb.History<SmartInsiderTransaction>(symbol, 300, Resolution.Daily);
foreach (SmartInsiderTransaction transaction in transactionHistory)
{
    Console.WriteLine($"{transaction.NoteText} at {transaction.EndTime}");
}

// Add Universe Selection for SmartInsiderIntention
IEnumerable<Symbol> IntentionSelection(IEnumerable<BaseData> altCoarse)
{
    return (from d in altCoarse.OfType<SmartInsiderIntentionUniverse>()
        where d.AmountValue.HasValue
        orderby d.AmountValue descending select d.Symbol).Take(10);
}
var intentionUniverse = qb.AddUniverse<SmartInsiderIntentionUniverse>(IntentionSelection);

// Historical Universe data
var intentionUniverseHistory = qb.UniverseHistory(intentionUniverse, qb.Time.AddDays(-10), qb.Time);
foreach (var intentions in intentionUniverseHistory)
{
    foreach (SmartInsiderIntentionUniverse intention in intentions)
    {
        Console.WriteLine($"{intention.Symbol.Value} amount at {intention.EndTime}: {intention.AmountValue}");
    }
}

// Add Universe Selection for SmartInsiderTransaction
IEnumerable<Symbol> TransactionSelection(IEnumerable<BaseData> altCoarse)
{
    return (from d in altCoarse.OfType<SmartInsiderTransactionUniverse>()
        where d.Amount.HasValue
        orderby d.Amount descending select d.Symbol).Take(10);
}
var transactionUniverse = qb.AddUniverse<SmartInsiderTransactionUniverse>(TransactionSelection);

// Historical Universe data
var transactionHistory = qb.UniverseHistory(transactionUniverse, qb.Time.AddDays(-10), qb.Time);
foreach (var transactions in transactionHistory)
{
    foreach (SmartInsiderTransactionUniverse transaction in transactions)
    {
        Console.WriteLine($"{transaction.Symbol.Value} amount at {transaction.EndTime}: {transaction.Amount}");
    }
}
```

Data Point Attributes

The Corporate Buybacks dataset provides `SmartInsiderIntention`, `SmartInsiderIntentionUniverse`, `SmartInsiderTransaction`, and `SmartInsiderTransactionUniverse` objects.

SmartInsiderIntention Attributes

`SmartInsiderIntention` objects have the following attributes:

SmartInsiderIntentionUniverse Attributes

`SmartInsiderIntentionUniverse` objects have the following attributes:

SmartInsiderTransaction Attributes

`SmartInsiderTransaction` objects have the following attributes:

SmartInsiderTransactionUniverse Attributes

`SmartInsiderTransactionUniverse` objects have the following attributes:

Datasets

Tiingo

Tiingo was founded by Rishi Singh in 2014. Tiingo goes beyond traditional news sources and focuses on finding rich, quality content written by knowledgeable writers. Their proprietary algorithms scan unstructured, non-traditional news and other information sources while tagging companies, topics, and assets. This refined system is backed by over ten years of research and development, and is written by former institutional quant traders. Because of this dedicated approach, Tiingo's News API is a trusted tool used by quant funds, hedge funds, pension funds, social media companies, and tech companies around the world.

Tiingo News Feed

Tiingo

Tiingo News Feed

Introduction

The Tiingo News Feed dataset by Tiingo tracks US Equity news releases. The data covers 10,000 US Equities, starts in January 2014, and is delivered on a second frequency. This dataset is creating by Tiingo integrating over 120 different news providers into their platform.

This dataset depends on the US Equity Security Master dataset because the US Equity Security Master dataset contains information on splits, dividends, and symbol changes.

For more information about the Tiingo News Feed dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

Tiingo was founded by Rishi Singh in 2014. Tiingo goes beyond traditional news sources and focuses on finding rich, quality content written by knowledgeable writers. Their proprietary algorithms scan unstructured, non-traditional news and other information sources while tagging companies, topics, and assets. This refined system is backed by over ten years of research and development, and is written by former institutional quant traders. Because of this dedicated approach, Tiingo's News API is a trusted tool used by quant funds, hedge funds, pension funds, social media companies, and tech companies around the world.

Getting Started

The following snippet demonstrates how to request data from the Tiingo News Feed dataset:

```C#
_aapl = AddEquity("AAPL", Resolution.Minute).Symbol;
_datasetSymbol = AddData<TiingoNews>(_aapl).Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 2014
Asset Coverage	10,000 US Equities
Data Density	Sparse
Resolution	Second
Timezone	UTC

Requesting Data

To add Tiingo News Feed data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```
C#
public class TiingoNewsDataAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2021, 1, 1);
        SetEndDate(2021, 6, 1);
        SetCash(100000);

        _symbol = AddEquity("AAPL", Resolution.Minute).Symbol;
        _datasetSymbol = AddData<TiingoNews>(_symbol).Symbol;
    }
}
```

Accessing Data

To get the current Tiingo News Feed data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```
C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var article = slice[_datasetSymbol];
        Log($"{_datasetSymbol} article description at {slice.Time}: {article.Description}");
    }
}
```

To iterate through all of the articles in the current `Slice` , call the `Get` method.

```
C#
public override void OnData(Slice slice)
{
    foreach (var kvp in slice.Get<TiingoNews>())
    {
        var datasetSymbol = kvp.Key;
        var article = kvp.Value;
        Log($"{datasetSymbol} article description at {slice.Time}: {article.Description}");
    }
}
```

Historical Data

To get historical Tiingo News Feed data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```
C#
var history = History<TiingoNews>(_datasetSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove a subscription, call the `RemoveSecurity` method.

```
C#
RemoveSecurity(_datasetSymbol);
```

If you subscribe to Tiingo News Feed data for assets in a dynamic universe, remove the dataset subscription when the asset leaves your universe. To view a common design pattern, see Track Security Changes .

Example Applications

The Tiingo News Feed enables you to accurately design strategies harnessing news articles on the companies you're trading. Examples include the following strategies:

- Creating a dictionary of sentiment scores for various words and assigning a sentiment score to the content of each news release
- Calculating the sentiment of news releases with Natural Language Processing (NLP)
- Trading securities when their news releases are tagged by Tiingo with current buzzwords
- Detecting impactful news in ETF constituents

Classic Algorithm Example

The following example algorithm assigns a sentiment score to each news article that's released for Apple. When the sentiment score is positive, the algorithm buys Apple stock. When the sentiment score is negative, it short sells Apple stock.

```C#
public class TiingoNewsDataAlgorithm : QCAlgorithm
{
    private Symbol _aapl;
    private Symbol _tiingoSymbol;
    private int _currentHoldings = 0;
    private int _targetHoldings = 0;
    // Custom word-score map to assign score for each word in article
    private Dictionary<string, int> _wordScores = new Dictionary<string, int>(){
        {"good", 1}, {"great", 1}, {"best", 1}, {"growth", 1},
        {"bad", -1}, {"terrible", -1}, {"worst", -1}, {"loss", -1}
    };

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Requesting Tiingo news data to obtain the updated news articles to calculate the sentiment score
        _aapl = AddEquity("AAPL", Resolution.Minute).Symbol;
        _tiingoSymbol = AddData<TiingoNews>(_aapl).Symbol;

        // Historical data
        var history = History<TiingoNews>(_tiingoSymbol, 14, Resolution.Daily);
        Debug($"We got {history.Count()} items from our history request");
    }

    public override void OnData(Slice slice)
    {
        if (slice.ContainsKey(_tiingoSymbol))
        {
            // Assign a sentiment score to the news article
            var titleWords = slice[_tiingoSymbol].Description.ToLower();
            var score = 0;
            foreach (KeyValuePair<string, int> entry in _wordScores)
            {
                if (titleWords.Contains(entry.Key))
                {
                    score += entry.Value;
                }
            }

            // Buy if aggregated sentiment score shows positive sentiment, sell vice versa
            if (score > 0)
            {
                _targetHoldings = 1;
            }
            else if (score < 0)
            {
                _targetHoldings = -1;
            }
        }

        // Buy or short sell if the sentiment has changed from our current holdings
        if (slice.ContainsKey(_aapl) && _currentHoldings != _targetHoldings)
        {
            SetHoldings(_aapl, _targetHoldings);
            _currentHoldings = _targetHoldings;
        }
    }
}
```

Framework Algorithm Example

The following example algorithm assigns a sentiment score to each news article that's released for Apple. When the sentiment score is positive, the algorithm buys Apple stock. When the sentiment score is negative, it short sells Apple stock. The algorithm holds positions for 14 days.

```C#
public class TiingoNewsDataAlgorithm : QCAlgorithm
{

    public override void Initialize()
    {
```

```
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var symbols = new[] {QuantConnect.Symbol.Create("AAPL", SecurityType.Equity, Market.USA)};
        AddUniverseSelection(new ManualUniverseSelectionModel(symbols));
        AddAlpha(new TiingoNewsAlphaModel());
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }

    public class TiingoNewsAlphaModel : AlphaModel
    {
        private Symbol _aapl;
        private Symbol _tiingoSymbol;
        private int _currentHoldings = 0;
        private int _targetHoldings = 0;
        // Custom word-score map to assign score for each word in article
        private Dictionary<string, int> _wordScores = new Dictionary<string, int>(){
            {"good", 1}, {"great", 1}, {"best", 1}, {"growth", 1},
            {"bad", -1}, {"terrible", -1}, {"worst", -1}, {"loss", -1}
        };

        public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
        {
            var insights = new List<Insight>();

            if (slice.ContainsKey(_tiingoSymbol))
            {
                // Assign a sentiment score to the news article by the word-score map
                var titleWords = slice[_tiingoSymbol].Description.ToLower();
                var score = 0;
                foreach (KeyValuePair<string, int> entry in _wordScores)
                {
                    if (titleWords.Contains(entry.Key))
                    {
                        score += entry.Value;
                    }
                }

                // Buy if aggregated sentiment score shows positive sentiment, sell vice versa
                if (score > 0)
                {
                    _targetHoldings = 1;
                }
                else if (score < 0)
                {
                    _targetHoldings = -1;
                }
            }

            // Buy or short sell if the sentiment has changed from our current holdings
            if (slice.ContainsKey(_aapl) && _currentHoldings != _targetHoldings)
            {
                _currentHoldings = _targetHoldings;
                var direction = _targetHoldings == 1 ? InsightDirection.Up : InsightDirection.Down;
                insights.Add(Insight.Price(_aapl, TimeSpan.FromDays(14), direction));
            }

            return insights;
        }

        public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
        {
            foreach (var security in changes.AddedSecurities)
            {
                _aapl = security.Symbol;

                // Requesting Tiingo news data to obtain the updated news articles to calculate the sentiment score
                _tiingoSymbol = algorithm.AddData<TiingoNews>(_aapl).Symbol;

                // Historical data
                var history = algorithm.History<TiingoNews>(_tiingoSymbol, 14, Resolution.Daily);
                algorithm.Debug($"We got {history.Count()} items from our history request");
            }
        }
    }
}
```

Data Point Attributes

The Tiingo News Feed dataset provides `TiingoNews` objects, which have the following attributes:

Datasets

Treasury Department

The Treasury Department is the executive agency responsible for promoting economic prosperity and ensuring the financial security of the United States. The Department is responsible for a wide range of activities such as advising the President on economic and financial issues, encouraging sustainable economic growth, and fostering improved governance in financial institutions. The Department of the Treasury operates and maintains systems that are critical to the nation's financial infrastructure, such as the production of coin and currency, the disbursement of payments to the American public, revenue collection, and the borrowing of funds necessary to run the federal government.

US Treasury Yield Curve

Treasury Department

US Treasury Yield Curve

Introduction

The US Treasury Yield Curve datasets tracks the yield curve rate from the US Department of the Treasury. The data starts in January 1990 and is delivered on a daily frequency. This dataset is calculated from composites of indicative, bid-side market quotations (not actual transactions) obtained by the Federal Reserve Bank of New York at or near 3:30 PM Eastern Time (ET) each trading day.

For more information about the US Treasury Yield Curve dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

The Treasury Department is the executive agency responsible for promoting economic prosperity and ensuring the financial security of the United States. The Department is responsible for a wide range of activities such as advising the President on economic and financial issues, encouraging sustainable economic growth, and fostering improved governance in financial institutions. The Department of the Treasury operates and maintains systems that are critical to the nation's financial infrastructure, such as the production of coin and currency, the disbursement of payments to the American public, revenue collection, and the borrowing of funds necessary to run the federal government.

Getting Started

The following snippet demonstrates how to request data from the US Treasury Yield Curve dataset:

```C#
_datasetSymbol = AddData<USTreasuryYieldCurveRate>("USTYCR").Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	January 1990
Coverage	1 Dataset
Data Density	Sparse
Resolution	Daily
Timezone	New York

Requesting Data

To add US Treasury Yield Curve data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```C#
public class USTreasuryYieldCurveDataAlgorithm : QCAlgorithm
{
    private Symbol _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2019, 1, 1);
        SetEndDate(2020, 6, 1);
        SetCash(100000);

        _datasetSymbol = AddData<USTreasuryYieldCurveRate>("USTYCR").Symbol;
    }
}
```

Accessing Data

To get the current US Treasury Yield Curve data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var dataPoint = slice[_datasetSymbol];
        Log($"{_datasetSymbol} one month value at {slice.Time}: {dataPoint.OneMonth}");
    }
}
```

Historical Data

To get historical US Treasury Yield Curve data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
var history = History<USTreasuryYieldCurveRate>(_datasetSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove your subscription to US Treasury Yield Curve data, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_datasetSymbol);
```

Example Applications

The US Treasury Yield Curve dataset enables you to monitor the yields of bonds with numerous maturities in your strategies. Examples include the following strategies:

- Short selling SPY when the yield curve inverts
- Buying short-term Treasuries and short selling long-term Treasuries when the yield curve becomes steeper (aka curve steepener trade)

Classic Algorithm Example

The following example algorithm short sells SPY for two years when the yield curve inverts:

```C#
public class USTreasuryDataAlgorithm : QCAlgorithm
{
    private Symbol _spySymbol;
    private Symbol _yieldCurveSymbol;
    private DateTime _lastInversion = DateTime.MinValue;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Request SPY as the market representative for trading
        _spySymbol = AddEquity("SPY", Resolution.Hour).Symbol;

        // Requesting yield curve data for trade signal generation (inversion)
        _yieldCurveSymbol = AddData<USTreasuryYieldCurveRate>("USTYCR").Symbol;

        // Historical data
        var history = History<USTreasuryYieldCurveRate>(_yieldCurveSymbol, 60, Resolution.Daily);
        Debug($"We got {history.Count()} items from our history request");
    }


    public override void OnData(Slice slice)
    {
        // Trade only based on updated yield curve data
        if (!slice.TryGetValue(_yieldCurveSymbol, out var rates))
        {
            return;
        }

        // Only advance if a year has gone by, since the inversion signal indicates longer term market regime that
will not revert in a short period
        if (Time - _lastInversion < TimeSpan.FromDays(365))
        {
            return;
        }

        // Normally, 10y yield should be greater than 2y yield due to default risk accumulation
        // But if an inversion occurs, it means the market expects a recession in short term such that the near-
expiry bond is more likely to default
        // if there is a yield curve inversion after not having one for a year, short sell SPY for two years for the
expected down market
        if (!Portfolio.Invested && rates.TwoYear > rates.TenYear)
        {
            Debug($"{Time} - Yield curve inversion! Shorting the market for two years");
            SetHoldings(_spySymbol, -0.5);
                _lastInversion = Time;
            return;
        }


        // If two years have passed, liquidate our position in SPY assuming the market starts resilience
        if (Time - _lastInversion >= TimeSpan.FromDays(365 * 2))
        {
            Liquidate(_spySymbol);
        }
    }
}
```

Framework Algorithm Example

The following example algorithm short sells SPY for two years when the yield curve inverts:

C#

```
public class USTreasuryYieldCurveDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        UniverseSettings.Resolution = Resolution.Hour;
        // Universe only have SPY as the market representative for trading
        var symbols = new[] {QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA)};
        AddUniverseSelection(new ManualUniverseSelectionModel(symbols));
        // Custom alpha model that emit insight according to US Treasury data
        AddAlpha(new USTreasuryAlphaModel(this));
        // Use insight weighting PCM to limit the size of investment, reduce the fluctuation of the portfolio value
        SetPortfolioConstruction(new InsightWeightingPortfolioConstructionModel((time) => null));
    }
}

public class USTreasuryAlphaModel : AlphaModel
{
    private Symbol? _spySymbol = null;
    private Symbol _yieldCurveSymbol;
    private DateTime _lastInversion = DateTime.MinValue;

    public USTreasuryAlphaModel(QCAlgorithm algorithm)
    {
        // Requesting yield curve data for trade signal generation (inversion)
        _yieldCurveSymbol = algorithm.AddData<USTreasuryYieldCurveRate>("USTYCR").Symbol;

        // Historical data
        var history = algorithm.History<USTreasuryYieldCurveRate>(_yieldCurveSymbol, 60, Resolution.Daily);
        algorithm.Debug($"We got {history.Count()} items from our history request");
    }

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = new List<Insight>();

        // Trade only based on updated yield curve data
        if (!slice.TryGetValue(_yieldCurveSymbol, out var rates) || _spySymbol == null)
        {
            return insights;
        }

        // Only advance if a year has gone by, since the inversion signal indicates longer term market regime that
will not revert in a short period
        if (slice.Time - _lastInversion < TimeSpan.FromDays(365))
        {
            return insights;
        }

        // Normally, 10y yield should be greater than 2y yield due to default risk accumulation
        // But if an inversion occurs, it means the market expects a recession in short term such that the near-
expiry bond is more likely to default
        // if there is a yield curve inversion after not having one for a year, short sell SPY for two years for the
expected down market
        if (!algorithm.Portfolio.Invested && rates.TwoYear > rates.TenYear)
        {
            algorithm.Debug($"{slice.Time} - Yield curve inversion! Shorting the market for two years");
            _lastInversion = slice.Time;
            insights.Add(Insight.Price(_spySymbol, slice.Time + TimeSpan.FromDays(2*365), InsightDirection.Down,
null, null, null, 0.5));
        }

        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            _spySymbol = security.Symbol;
        }
    }
}
```

Data Point Attributes

The US Treasury Yield Curve dataset provides `USTreasuryYieldCurveRate` objects, which have the following attributes:

Datasets

VIX Central

VIX Central was founded by Eli Mintz in 2012 with goal of displaying historical VIX term structures in a simple and intuitive interface. VIX Central provides access to real-time and historical VIX data for individual investors.

VIX Central Contango

VIX Central

VIX Central Contango

Introduction

The VIX Central Contango dataset by VIX Central tracks VIX Futures (VX) contango data. The data covers 12 Futures contracts closest to expiry/maturity, starts in June 2010, and is delivered on a daily frequency. The dataset is created by QuantConnect downloading data from VIX Central website, which collects and analyses VIX and VX (VIX Futures) data.

Contango and Backwardation are terms used to describe if participants in the Futures market are overpaying or underpaying relative to the "spot" price of the underlying commodity when trading a Futures contract ("spot" price is the price of the actual commodity/asset at a given moment in time). Contango and backwardation can be used to determine forward-looking expectations of the commodity's spot price by the time the Future has expired/matured and is set to be delivered by participants of the Futures market. As Futures near their expiration/maturity date, contango and backwardation curves tend to converge on the spot price of the commodity at the time of expiration.

For more information about the VIX Central Contango dataset, including CLI commands and pricing, see the dataset listing .

About the Provider

VIX Central was founded by Eli Mintz in 2012 with goal of displaying historical VIX term structures in a simple and intuitive interface. VIX Central provides access to real-time and historical VIX data for individual investors.

Getting Started

The following snippet demonstrates how to request data from the VIX Central Contango dataset:

```
C#
_datasetSymbol = AddData<VIXCentralContango>("VIX", Resolution.Daily).Symbol;
```

Data Summary

The following table describes the dataset properties:

Property	Value
Start Date	June 2010
Asset Coverage	1 Futures Chain with 12 contracts
Data Density	Regular
Resolution	Daily
Timezone	New York

Requesting Data

To add VIX Central Contango data to your algorithm, call the `AddData` method. Save a reference to the dataset `Symbol` so you can access the data later in your algorithm.

```C#
public class VixCentralContangoAlgorithm : QCAlgorithm
{
    private Symbol _datasetSymbol;

    public override void Initialize()
    {
        SetStartDate(2014, 1, 1);
        SetEndDate(2018, 1, 1);
        SetCash(25000);

        _datasetSymbol = AddData<VIXCentralContango>("VX", Resolution.Daily).Symbol;
    }
}
```

Accessing Data

To get the current VIX Central Contango data, index the current `Slice` with the dataset `Symbol` . Slice objects deliver unique events to your algorithm as they happen, but the `Slice` may not contain data for your dataset at every time step. To avoid issues, check if the `Slice` contains the data you want before you index it.

```C#
public override void OnData(Slice slice)
{
    if (slice.ContainsKey(_datasetSymbol))
    {
        var dataPoint = slice[_datasetSymbol];
        Log($"{_datasetSymbol} front month at {slice.Time}: {dataPoint.FrontMonth}");
    }
}
```

Historical Data

To get historical VIX Central Contango data, call the `History` method with the dataset `Symbol` . If there is no data in the period you request, the history result is empty.

```C#
var history = History<VIXCentralContango>(_datasetSymbol, 100, Resolution.Daily);
```

For more information about historical data, see History Requests .

Remove Subscriptions

To remove your subscription to VIX Central Contango data, call the `RemoveSecurity` method.

```C#
RemoveSecurity(_datasetSymbol);
```

Example Applications

The VIX Central Contango dataset enables you to explore VIX Future contracts pricing data. Examples include the following strategies:

- Determining forward-looking expectations by Futures market participants of the underlying commodity's spot price by the time of expiration/maturity
- Creating cash-and-carry arbitrage strategies by trading the spread/convergence of the contango/backwardation curves as a Future nears expiration/maturity

Classic Algorithm Design

The following example algorithm buys SPY when the percentage change between contract F2 and F1 (`Contango_F2_Minus_F1`) is positive. Otherwise, it remains in cash.

```
C#
public class VixCentralContangoAlgorithm : QCAlgorithm
{
    private Symbol _spy, _contango;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(25000);
        // SPY as market representative for macroeconomy indicator trading
        _spy = AddEquity("SPY", Resolution.Daily).Symbol;
        // Request VIX Contango data for trade signal generation
        _contango = AddData<VIXCentralContango>("VX", Resolution.Daily).Symbol;
    }

    public override void OnData(Slice slice)
    {
        // Trade base on the updated VIX contango data
        if (!slice.ContainsKey(_contango))
        {
            return;
        }
        var contangoData = slice.Get<VIXCentralContango>(_contango);
        var ratio = contangoData?.Contango_F2_Minus_F1 ?? 0;

        // Bet the market will go up if F2 - F1 price percentage change is positive, meaning the near-term market
volatility is expected to be low
        if (!Portfolio.Invested && ratio > 0)
        {
            SetHoldings(_spy, 1);
        }
        // Liqudiate otherwise to avoid excessive market volatility
        else if(ratio < 0)
        {
            Liquidate();
        }
    }
}
```

Framework Algorithm Design

The following example algorithm buys SPY when the percentage change between contract F2 and F1 (`Contango_F2_Minus_F1`) is positive. Otherwise, it remains in cash.

```
C#
public class VixCentralContangoAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(25000);

        UniverseSettings.Resolution = Resolution.Daily;
        // Include SPY as market representative for macroeconomy indicator trading
        var symbols = new[] {QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA)};
        AddUniverseSelection(new ManualUniverseSelectionModel(symbols));
        // Custom alpha model that emit insights base on VIX contango data
        AddAlpha(new ContangoAlphaModel(this));
        // Equal weighting investment to evenly dissipate the capital concentration risk on non-systematic risky
event
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }
}

public class ContangoAlphaModel : AlphaModel
{
    private Symbol _contango;
    private List<Symbol> _symbols;
```

```
    public ContangoAlphaModel(QCAlgorithm algorithm)
    {
        _symbols = new List<Symbol>();
        // Request VIX Contango data for trade signal generation
        _contango = algorithm.AddData<VIXCentralContango>("VX", Resolution.Daily).Symbol;

        // Historical data
        var history = algorithm.History<VIXCentralContango>(_contango, 60, Resolution.Daily);
        algorithm.Debug($"We got {history.Count()} items from our history request");
    }

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = new List<Insight>();

        // Trade base on the updated VIX contango data
        if (!slice.ContainsKey(_contango))
        {
            return insights;
        }
        var contangoData = slice.Get<VIXCentralContango>(_contango);
        var ratio = contangoData?.Contango_F2_Minus_F1 ?? 0;

        // Bet the market will go up if F2 - F1 price percentage change is positive, meaning the near-term market
volatility is expected to be low
        if (!algorithm.Portfolio.Invested && ratio > 0)
        {
            foreach (var symbol in _symbols)
            {
                insights.Add(Insight.Price(symbol, Expiry.OneMonth, InsightDirection.Up));
            }
        }
        // Liqudiate otherwise to avoid excessive market volatility
        else if (ratio < 0)
        {
            foreach (var symbol in _symbols)
            {
                insights.Add(Insight.Price(symbol, Expiry.OneMonth, InsightDirection.Flat));
            }
        }

        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach (var symbol in changes.RemovedSecurities.Select(x=> x.Symbol))
        {
            _symbols.Remove(symbol);
        }
        _symbols.AddRange(changes.AddedSecurities.Select(x=> x.Symbol));
    }
}
```

Data Point Attributes

The VIX Central Contango dataset provides `VIXCentralContango` objects, which have the following attributes:

Importing Data

Importing Data

Key Concepts

Introduction

Custom data is your own external data that's not from the Dataset Market. You can use custom data to inform trading decisions and to simulate trades on unsupported securities. To get custom data into your algorithms, you download the entire file at once or read it line-by-line with a custom data reader. If you use a custom data reader, LEAN sends the data to the `OnData` method in your algorithm.

File Providers

The most common file providers to use are the Object Store , Dropbox, GitHub, and Google Sheets.

Object Store

The Object Store is the fastest file provider. If you import files from remote providers, you will be restricted by their rate limits and your download speed.

Dropbox

If you store your custom data in Dropbox, you need to create a link to the file and add `?dl=1` to the end of the file URL. To create file links, see How to share files or folders in the Dropbox documentation. The `?dl=1` parameter lets you download the direct file link, not the HTML page of the file.

GitHub

If you store your custom data in GitHub, you can use public or private repositories. When you download the data, use the raw file (for example, https://raw.githubusercontent.com/<organization>/<repo>/<path>). For instructions on accessing the raw file, see Viewing or copying the raw file content in the GitHub documentation.

Google Sheets

If you store your custom data in Google Sheets, you need to create a link to the file. To create file links, see Make Google Docs, Sheets, Slides & Forms public in the Google Docs documentation. Choose the CSV format to find `/pub?gid={SHEET_ID}&single=true&output=csv` parameter on the end of the link. Google Sheets don't support exporting data in JSON format.

Stream Custom Data

To receive your custom data in the `OnData` method, create a custom type and then create a data subscription. The custom data type tells LEAN where to get your data and how to read it.

All custom data types must extend the `BaseData` class and override the `GetSource` and `Reader` methods

```C#
public class MyCustomDataType : BaseData
{
    public override DateTime EndTime { get; set; }
    public decimal Property1 { get; set; } = 0;

    public override SubscriptionDataSource GetSource(
        SubscriptionDataConfig config,
        DateTime date,
        bool isLive)
    {
        return new SubscriptionDataSource("<sourceURL>", SubscriptionTransportMedium.RemoteFile);
    }

    public override BaseData Reader(
        SubscriptionDataConfig config,
        string line,
        DateTime date,
        bool isLive)
    {
        if (string.IsNullOrWhiteSpace(line.Trim()) || char.IsDigit(line[0]))
        {
            return null;
        }

        var data = line.Split(',');
        return new MyCustomDataType()
        {
            Time = DateTime.ParseExact(data[0], "yyyyMMdd", CultureInfo.InvariantCulture),
            EndTime = Time.AddDays(1),
            Symbol = config.Symbol,
            Value = data[1].IfNotNullOrEmpty(
                s => decimal.Parse(s, NumberStyles.Any, CultureInfo.InvariantCulture)),
            Property1 = data[2].IfNotNullOrEmpty(
                s => decimal.Parse(s, NumberStyles.Any, CultureInfo.InvariantCulture))
        };
    }
}
```

For more information about custom data types, see Streaming Data .

Download Bulk Data

The `Download` method downloads the content served from a local file or URL and then returns it as a string.

```C#
var content = Download("<filePathOrURL>");
```

For more information about bulk downloads, see Bulk Downloads .

File Quotas

There are no limits to the number of files you can load from the Object Store during a single backtest or Research Environment session in QuantConnect Cloud.

The following table shows the number of remote files you can download during a single backtest or Research Environment session in QuantConnect Cloud:

Organization Tier	File Quota
Free	25
Quant Researcher	100
Team	250
Trading Firm	1,000
Institutional	Unlimited

Remote files can be up to 200 MB in size and can have names up to 200 characters long.

Rate Limits

We do not impose a rate limit on file downloads but often external providers do. Dropbox caps download speeds to 10 kb/s after 3-4 download requests. To ensure your algorithms run fast, only use a small number of small custom data files or use the Object Store.

Timeouts

In cloud algorithms, the download methods have a 10-second timeout period. If the methods don't download the data within 10 seconds, LEAN throws an error.

Importing Data

Streaming Data

Streaming Data

Key Concepts

Introduction

There are two techniques to import data into your algorithm. You can either manually import the entire file or stream the file line-by-line into your algorithm's `OnData` event. This page explores streaming a file's contents into your algorithm line-by-line. The data you import can be from a remote server or the Object Store .

Data Formats

Common data formats are CSV , JSON , XML , and ZIP but you can use any file type that can be read over the internet. For Excel files, double check the raw data format for parsing in the data reader, since data will be formatted for convenient visualization in Excel application view. To avoid confusion of data format, save the spreadsheet as a CSV file and open it in a text editor to confirm the raw data format.

If you import from a remote file provider , each request has a one-second overhead, so package your custom data to minimize requests. Bundle dates together where possible to speed up execution. The Object Store file provider gives you the fastest execution because you don't need to download the files on every execution.

Point-In-Time Tickers

Tickers are a string shortcode representation for an asset. Some examples of popular tickers include "AAPL" for Apple Corporation and "IBM" for International Business Machines Corporation. These tickers often change when the company rebrands or they undergo a merger or reverse merger.

The ticker of an asset is not the same as the `Symbol` . `Symbol` objects are permanent and track the underlying entity. When a company rebrands or changes its name, the `Symbol` object remains constant, giving algorithms a way to reliably track assets over time.

Tickers are also often reused by different brokerages. For example Coinbase, a leading US Crypto Brokerage lists the "BTCUSD" ticker for trading. Its competitor, Bitfinex, also lists "BTCUSD". You can trade both tickers with LEAN. `Symbol` objects allow LEAN to identify which market you reference in your algorithms.

To create a `Symbol` object for a point-in-time ticker, call the `GenerateEquity` method to create the security identifier and then call the `Symbol` constructor. For example, Heliogen, Inc. changed their ticker from ATHN to HLGN on December 31, 2021. To convert the ATHN ticker to the Equity `Symbol` , type:

```C#
// Generate an Equity security identifier for a ticker and then create a symbol with it.
var ticker = "ATHN";
var securityID = SecurityIdentifier.GenerateEquity(ticker, Market.USA, mappingResolveDate:new DateTime(2021, 12, 1));
var symbol = new Symbol(securityID, ticker);
```

In the preceding code snippet, the `mappingResolveDate` must be a date when the point-in-time ticker was trading.

For examples of point-in-time tickers in custom data types, refer to the CSV Format Example and JSON Format Example .

Set Data Sources

The `GetSource` method in your custom data class instructs LEAN where to find the data.

```C#
public class MyCustomDataType : BaseData
{
    public override SubscriptionDataSource GetSource(
        SubscriptionDataConfig config,
        DateTime date,
        bool isLiveMode)
    {
        if (isLiveMode)
        {
            return new SubscriptionDataSource("https://www.bitstamp.net/api/ticker/",
SubscriptionTransportMedium.Rest);
        }

        var source = $"http://my-ftp-server.com/{config.Symbol.Value}/{date:yyyyMMdd}.csv";
        return new SubscriptionDataSource(source, SubscriptionTransportMedium.RemoteFile);

        /*
        // Example of loading from the Object Store:
        return new SubscriptionDataSource(Bitstamp.KEY,
            SubscriptionTransportMedium.ObjectStore);

        // Example of loading a remote zip file:
        return new SubscriptionDataSource(
            "https://cdn.quantconnect.com/uploads/multi_csv_zipped_file.zip",
            SubscriptionTransportMedium.RemoteFile,
            FileFormat.ZipEntryName);

        // Example of loading a remote zip file and accessing a CSV file inside it:
        return new SubscriptionDataSource(
            "https://cdn.quantconnect.com/uploads/multi_csv_zipped_file.zip#csv_file_10.csv",
            SubscriptionTransportMedium.RemoteFile,
            FileFormat.ZipEntryName);
        */
    }
}
```

The following table describes the arguments the `GetSource` method accepts:

Argument	Data Type	Description
`config`	`SubscriptionDataConfig`	The subscription configuration
`date`	`DateTime`	Date of this source file
`isLiveMode`	`bool`	`true` if algorithm is running in live mode

You can use these arguments to create `SubscriptionDataSource` objects representing different locations and formats. The following table describes the arguments the `SubscriptionDataSource` accepts:

Argument	Data Type	Description	Default Value
`source`	`string`	Data source location	
`transportMedium`	`SubscriptionTransportMedium`	The transport medium to be used to retrieve data from the source	
`format`	`FileFormat`	The format of the data within the source	`FileFormat.Csv`
`headers`	`IEnumerable<KeyValuePair<string, string>>`	The headers to be used for this source. In cloud algorithms, each of the key-value pairs can consist of up to 1,000 characters.	`null`

The `FileFormat` enumeration has the following members:

The `SubscriptionTransportMedium` enumeration has the following members:

Member	Description	Example
`LocalFile`	The data comes from disk	Lean.DataSource.CBOE
`RemoteFile`	The data is downloaded from a remote source	`Custom Securities Examples`
`Rest`	The data comes from a rest call that is polled and returns a single line/data point of information	`LiveMode` case of Demonstration Algorithm
`ObjectStore`	The data comes from the object store	Example of Custom Data

Parse Custom Data

The `Reader` method of your custom data class takes one line of data from the source location and parses it into one of your custom objects. You can add as many properties to your custom data objects as you need, but the following table describes the properties you must set. When there is no useable data in a line, the method should return `null` . LEAN repeatedly calls the `Reader` method until the date/time advances or it reaches the end of the file.

Property	Description
`Symbol`	You can set this property to `config. Symbol` .
`Time`	The time when the data sample starts.
`EndTime`	The time when the data sample ends and when LEAN should add the sample to a Slice .
`Value`	The default data point value (`decimal`).

The following table describes the arguments the `Reader` method accepts:

Argument	Data Type	Description
`config`	`SubscriptionDataConfig`	The subscription configuration
`line`	`string`	Content from the requested data source
`date`	`DateTime`	Date of this source file
`isLiveMode`	`bool`	`true` if algorithm is running in live mode

You can use these arguments to create `BaseData` objects from different sources.

```C#
public class MyCustomDataType : BaseData
{
    public override BaseData Reader(
        SubscriptionDataConfig config,
        string line,
        DateTime date,
        bool isLiveMode)
    {
        if (string.IsNullOrWhiteSpace(line.Trim()))
        {
            return null;
        }

        if (isLiveMode)
        {
            var custom = JsonConvert.DeserializeObject<MyCustomDataType>(line);
            custom.EndTime = DateTime.UtcNow.ConvertFromUtc(config.ExchangeTimeZone);
            return custom;
        }

        if (!char.IsDigit(line[0]))
        {
            return null;
        }

        var data = line.Split(',');
        return new MyCustomDataType()
        {
            Time = DateTime.ParseExact(data[0], "yyyyMMdd", CultureInfo.InvariantCulture),
            EndTime = Time.AddDays(1),
            Symbol = config.Symbol,
            Value = data[1].IfNotNullOrEmpty(
                s => decimal.Parse(s, NumberStyles.Any, CultureInfo.InvariantCulture)),
        };
    }
}
```

Unsorted Data

By default, LEAN expects the data in chronological order. If the data is unsorted, set the `Sort` property of the `SubscriptionDataSource` object to `true` .

```C#
public class MyCustomDataType : BaseData
{
    public override SubscriptionDataSource GetSource(
        SubscriptionDataConfig config,
        DateTime date,
        bool isLiveMode)
    {
        var source = $"http://my-ftp-server.com/{config.Symbol.Value}/{date:yyyyMMdd}.csv";
        return new SubscriptionDataSource(source, SubscriptionTransportMedium.RemoteFile)
        {
            Sort = true
        };
    }
}
```

LEAN uses `EndTime` property of your custom data to sort it.

Object Store

The Object Store file provider gives you the fastest execution because you don't need to download the files on every execution. To upload the data files into the Object Store, use the Algorithm Lab , CLI , or Research Environment . To then pull the data from the Object Store into an algorithm, set the custom data source to read from the file in Object Store and parse the data in the `Reader` method.

```C#
public class MyCustomDataType : BaseData
{
    public override SubscriptionDataSource GetSource(
        SubscriptionDataConfig config,
        DateTime date,
        bool isLiveMode)
    {
        return new SubscriptionDataSource("<YourCSVKey>", SubscriptionTransportMedium.ObjectStore, FileFormat.Csv);
        // return new SubscriptionDataSource("<YourJSONKey>", SubscriptionTransportMedium.ObjectStore,
FileFormat.UnfoldingCollection);
    }
}
```

Set Properties

To set the Symbol Properties of the custom data, provide a `SymbolProperties` object when you subscribe to the dataset. The ticker you pass to the `SymbolProperties` constructor and the `AddData` `add_data` method must be the same.

To set the Exchange Hours of the custom data, provide a `SecurityExchangeHours` object when you subscribe to the dataset. The default hours are for the market to be open 24/7.

```C#
var ticker = "ABC";
var properties = new SymbolProperties("description", "USD", 1, 0.01, 0.01, ticker);
var hours = MarketHoursDatabase.GetEntry(Market.USA, "SPY", SecurityType.Equity).ExchangeHours;
AddData<CustomData>(ticker, properties, hours);
```

Live Trading Considerations

In live trading, we pass custom data to your algorithm as soon as it arrives. The time it arrives may not align with the time of other slices . Design your algorithm to handle unsychronized data so that you don't run into issues.

Demonstration Algorithm

The following example algorithm implements a custom data source for the Bitstamp API.

```C#
```

```c#
using Newtonsoft.Json;

public class CustomDataBitstampAlgorithm : QCAlgorithm
{
    private Symbol _customDataSymbol;

    public override void Initialize()
    {
        SetStartDate(2020, 9, 1);
        SetEndDate(2020, 12, 31);
        // Define the symbol and "type" of our custom data.
        _customDataSymbol = AddData<Bitstamp>("BTC", Resolution.Daily).Symbol;
        // Get some historical data.
        var history = History<Bitstamp>(_customDataSymbol, 200, Resolution.Daily);
    }

    // Get the data of the current day.
    public void OnData(Bitstamp data)
    {
        Plot(_customDataSymbol, "Price", data.Close);
    }

    public class Bitstamp : BaseData
    {
        public int Timestamp = 0;
        public decimal Open = 0, High = 0, Low = 0, Close = 0, Bid = 0, Ask = 0, WeightedPrice = 0, VolumeBTC = 0,
VolumeUSD = 0;

        public override SubscriptionDataSource GetSource(SubscriptionDataConfig config, DateTime date, bool
isLiveMode)
        {
            if (isLiveMode)
            {
                return new SubscriptionDataSource("https://www.bitstamp.net/api/ticker/",
SubscriptionTransportMedium.Rest);
            }
            var source =
"https://raw.githubusercontent.com/QuantConnect/Documentation/master/Resources/datasets/custom-data/bitstampusd.csv";
            return new SubscriptionDataSource(source, SubscriptionTransportMedium.RemoteFile);
        }

        public override BaseData Reader(SubscriptionDataConfig config, string line, DateTime date, bool isLiveMode)
        {
            if (string.IsNullOrWhiteSpace(line.Trim()))
            {
                return null;
            }
            var coin = new Bitstamp() {Symbol = config.Symbol};
            // In live trading, parse the JSON file.
            if (isLiveMode)
            {
                //Example Line Format:
                //{"high": "441.00", "last": "421.86", "timestamp": "1411606877", "bid": "421.96", "vwap": "428.58",
"volume": "14120.40683975", "low": "418.83", "ask": "421.99"}
                coin = JsonConvert.DeserializeObject<Bitstamp>(line);
                coin.EndTime = DateTime.UtcNow.ConvertFromUtc(config.ExchangeTimeZone);
                coin.Time = coin.EndTime.AddDays(-1);
                coin.Value = coin.Close;
                return coin;
            }

            // In backtests, parse the CSV file.
            //Example Line Format:
            //Date      Open   High    Low     Close   Volume (BTC)    Volume (Currency)   Weighted Price
            //2011-09-13 5.8    6.0     5.65    5.97    58.37138238,    346.0973893944      5.929230648356
            if (!char.IsDigit(line[0]))
            {
                return null;
            }
            var data = line.Split(',');
            // If value is zero, return null.
            coin.Value = data[4].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
            if (coin.Value == 0)
            {
                return null;
            }
            coin.Time = DateTime.Parse(data[0], CultureInfo.InvariantCulture);
            coin.EndTime = coin.Time.AddDays(1);
            coin.Open = data[1].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
```

```
CultureInfo.InvariantCulture));
            coin.High = data[2].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
            coin.Low = data[3].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
            coin.VolumeBTC = data[5].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
            coin.VolumeUSD = data[6].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
            coin.WeightedPrice = data[7].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
            coin.Close = coin.Value;
            return coin;
        }
    }
}
```

Streaming Data

Custom Securities

Custom Securities

Key Concepts

Introduction

To receive your custom data in the `OnData` method instead of in a bulk download, create a custom type and then create a data subscription. The custom data type tells LEAN where to get your data and how to read it. All custom data types must extend from `BaseData` and override the `Reader` and `GetSource` methods.

Unlike custom data, native QC data gets the full benefit of security modeling like splits and dividends. Custom data is ignorant of everything about the actual market. Things like market hours, delistings, and mergers don't work with custom data.

Create Subscriptions

After you define the custom data class, in the `Initialize` method of your algorithm, call the `AddData` method. This method gives LEAN the type factory to create the objects, the name of the data, and the resolution at which to poll the data source for updates.

```C#
public class MyAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    public override void Initialize()
    {
        _symbol = AddData<MyCustomDataType>("<name>", Resolution.Daily).Symbol;
    }
}
```

The `resolution` argument should match the resolution of your custom dataset. The lowest reasonable resolution is every minute. Anything more frequent than every minute is very slow to execute. The frequency that LEAN checks the data source depends on the `resolution` argument. The following table shows the polling frequency of each resolution:

Resolution	Update Frequency
Daily	Every 30 minutes
Hour	Every 30 minutes
Minute	Every minute
Second	Every second
Tick	Constantly checks for new data

There are several other signatures for the `AddData` method.

```
AddData<T>(ticker, resolution);
AddData<T>(ticker, resolution, fillForward, leverage);
AddData<T>(ticker, resolution, timeZone, fillForward, leverage);
AddData<T>(ticker, properties, exchangeHours, resolution, fillForward, leverage);
```

Receive Custom Data

As your data reader reads your custom data file, LEAN adds the data points in the `Slice` it passes to your algorithm's `OnData` method. To collect the custom data, use the `Symbol` or name of your custom data subscription. You can access the `Value` and custom properties of your custom data class from the `Slice` . To access the custom properties, use the custom attribute .

C#

```
public class MyAlgorithm : QCAlgorithm
{
    public override void OnData(Slice slice)
    {
        if (slice.ContainsKey(_symbol))
        {
            var customData = slice[_symbol];
            var value = customData.Value;
            var property1 = customData.Property1;
        }
    }

    // You can also get the data directly with OnData(<dataClass>) method
    public void OnData(MyCustomDataType slice)
    {
        var value = slice.Value;
        var property1 = slice.Property1;
    }
}
```

Debugging

In backtests, you can use the IDE debugging capabilities to inspect the variables to understand the internal state of the custom data implementation. For more information about backtest debugging, see the Debugging documentation for Cloud Platform , Local Platform , or the CLI .

In live trading, consider adding logging statements to help with debugging since the debugger isn't available. To log information from inside your custom data class, set a class member that references the algorithm.

```C#
public class MyCustomDataTypeAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        MyCustomDataType.ALGORITHM = this;
        var symbol = AddData<MyCustomDataType>("<name>", Resolution.Daily).Symbol;
    }
}

public class MyCustomDataType : BaseData
{
    public static QCAlgorithm ALGORITHM;

    public override BaseData Reader(SubscriptionDataConfig config, string line, DateTime date, bool isLiveMode)
    {
        if (!char.IsDigit(line.Trim()[0]))
        {
            // Display the line with the header
            ALGORITHM.Debug($"HEADER: {line}");
            return null;
        }
        var data = line.Split(',');
        return new MyCustomDataType()
        {
            Time = DateTime.ParseExact(data[0], "yyyyMMdd", CultureInfo.InvariantCulture),
            EndTime = Time.AddDays(1),
            Symbol = config.Symbol,
            Value = data[1].IfNotNullOrEmpty(
                s => decimal.Parse(s, NumberStyles.Any, CultureInfo.InvariantCulture)),
        };
    }
}
```

Set the Benchmark

To set your custom data source as the benchmark, in the `Initialize` method, call the `SetBenchmark` method with the `Symbol` of your custom data subscription.

```C#
var symbol = AddData<MyCustomDataType>("<name>", Resolution.Daily).Symbol;
SetBenchmark(symbol);
```

Avoid Look-Ahead Bias

Look-ahead bias occurs when you make decisions with information that wouldn't be available until some time in the future. In backtesting, look-ahead bias occurs when you receive data earlier than it would actually be available in reality. If look-ahead bias seeps into your algorithm, it will perform differently between live trading and backtesting.

To avoid look-ahead bias, set the timestamp of data points to the time when the data would actually be available. A lot of external sources apply timestamps to data differently than we do. For instance, on some platforms, daily bars are displayed with the date that they opened. We display daily bars with the date they closed. If you set the `EndTime` to the start of the bar, you'll receive the bar earlier in backtests than you would in live trading.

Time Modeling

Data types classes in LEAN inherit from the `BaseData` class that defines the `Time` and `EndTime` properties. These properties represent the time period of which the data was built. If the data type occurs in a singular point in time, they have no period, so `Time` and `EndTime` are the same. Regardless of the period, LEAN uses the time when the data sample ends, `EndTime` , to add the sample to a `Slice` .

Historical Data

To get a list of historical data points for a custom dataset, call the `History< customDatasetClass >` method with the dataset `Symbol` . For more information about historical data, see Data Points .

Market Hours

The default market hours for custom securities is to be always open. To set the market hours, pass an `exchangeHours` argument to the `AddData` method.

```
C#
var symbol = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
var symbolProperties = SymbolPropertiesDatabase.GetSymbolProperties(symbol.ID.Market, symbol, SecurityType.Equity,
"USD");
var securityExchangeHours = MarketHoursDatabase.GetExchangeHours(symbol.ID.Market, symbol, SecurityType.Equity);
var security = AddData<CustomType>(symbol, symbolProperties, securityExchangeHours, Resolution.Hour);
```

Custom Securities

CSV Format Example

Introduction

This page explains how to import custom data of a single security sourced in CSV format.

Data Format

The following snippet shows an example CSV file:

```
Date,Open,High,Low,Close,SharesTraded,Tur11er(Rs.Cr)
1997-01-01,905.2,941.4,905.2,939.55,38948210,978.21
1997-01-02,941.95,944,925.05,927.05,49118380,1150.42
1997-01-03,924.3,932.6,919.55,931.65,35263845,866.74
...
2014-07-24,7796.25,7835.65,7771.65,7830.6,117608370,6271.45
2014-07-25,7828.2,7840.95,7748.6,7790.45,153936037,7827.61
2014-07-28,7792.9,7799.9,7722.65,7748.7,116534670,6107.78
```

The data in the file must be in chronological order.

Define Custom Types

To define a custom data type, inherit the `BaseData` class and override the GetSource and Reader methods.

C#

```
public class MyCustomDataType : BaseData
{
    public decimal Open;
    public decimal High;
    public decimal Low;
    public decimal Close;

    public override SubscriptionDataSource GetSource(
        SubscriptionDataConfig config,
        DateTime date,
        bool isLiveMode)
    {
        if (!isLiveMode)
        {
            return new SubscriptionDataSource("<CustomDataKey>", SubscriptionTransportMedium.ObjectStore,
FileFormat.Csv);
        }
        return new
SubscriptionDataSource("https://raw.githubusercontent.com/QuantConnect/Documentation/master/Resources/datasets/custom-
data/csv-data-example.csv", SubscriptionTransportMedium.RemoteFile);
    }

    public override BaseData Reader(
        SubscriptionDataConfig config,
        string line,
        DateTime date,
        bool isLiveMode)
    {
        if (string.IsNullOrWhiteSpace(line.Trim()))
        {
            return null;
        }

        var index = new MyCustomDataType();

        try
        {
            var data = line.Split(',');
            index.Symbol = config.Symbol;
            index.Time = DateTime.Parse(data[0], CultureInfo.InvariantCulture);
            index.EndTime = index.Time.AddDays(1);
            index.Open = data[1].IfNotNullOrEmpty<decimal>(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
            index.High = data[2].IfNotNullOrEmpty<decimal>(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
            index.Low = data[3].IfNotNullOrEmpty<decimal>(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
            index.Close = data[4].IfNotNullOrEmpty<decimal>(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
            index.Value = index.Close;
        }
        catch
        {
        }
        return index;
    }
}
```

Create Subscriptions

To create a subscription for the custom type, in the `Initialize` method, call the `AddData` method. The `AddData` method returns a `Security` object, which contains a `Symbol` . Save a reference to the `Symbol` so you can use it in `OnData` to access the security data in the `Slice` .

```C#
public class MyAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    public override void Initialize()
    {
        _symbol = AddData<MyCustomDataType>("MyCustomDataType", Resolution.Daily).Symbol;
    }
}
```

The `AddData` method creates a subscription for a single custom data asset and adds it to your user-defined universe.

Receive Custom Data

As your data reader reads your custom data file, LEAN adds the data points in the `Slice` it passes to your algorithm's `OnData` method. To collect the custom data, use the `Symbol` or name of your custom data subscription. You can access the `Value` and custom properties of your custom data class from the `Slice` . To access the custom properties, use the custom attribute .

```C#
public class MyAlgorithm : QCAlgorithm
{
    public override void OnData(Slice slice)
    {
        if (slice.ContainsKey(_symbol))
        {
            var customData = slice[_symbol];
            var close = customData.Close;
        }
    }

    // You can also get the data directly with OnData(<dataClass>) method
    public void OnData(MyCustomDataType customData)
    {
        var close = customData.Close;
    }
}
```

Demonstration Algorithms

BubbleAlgorithm.cs C# CustomDataNIFTYAlgorithm.cs C# DescendingCustomDataObjectStoreRegressionAlgorithm.py Python

Custom Securities

JSON Format Example

Introduction

This page explains how to import custom data of a single security sourced in JSON format.

Data Format

The following code snippet shows an example JSON file

```
[
    {
      "Date": "1997-01-01",
      "Open": 905.2,
      "High": 941.4,
      "Low": 905.2,
      "Close": 939.55,
      "SharesTraded": 38948210,
      "Tur11er(Rs.Cr)": 978.21
    },
    {
      "Date": "1997-01-02",
      "Open": 941.95,
      "High": 944,
      "Low": 925.05,
      "Close": 927.05,
      "SharesTraded": 49118380,
      "Tur11er(Rs.Cr)": 1150.42
    },
    ...
    {
      "Date": "2014-07-25",
      "Open": 7828.2,
      "High": 7840.95,
      "Low": 7748.6,
      "Close": 7790.45,
      "SharesTraded": 153936037,
      "Tur11er(Rs.Cr)": 7827.61
    },
    {
      "Date": "2014-07-28",
      "Open": 7792.9,
      "High": 7799.9,
      "Low": 7722.65,
      "Close": 7748.7,
      "SharesTraded": 116534670,
      "Tur11er(Rs.Cr)": 6107.78
    }
]
```

The data in the file must be in chronological order.

Define Custom Types

To define a custom data type, inherit the `BaseData` class and override the GetSource and Reader methods.

If you need to create multiple objects in your `Reader` method from a single `line` , follow these steps:

1. In the `GetSource` method, pass `FileFormat.UnfoldingCollection` as the third argument to the `SubscriptionDataSource` constructor.
2. In the `Reader` method, order the objects by their timestamp and then return a

`BaseDataCollection(endTime, config.Symbol, objects)` where `objects` is a list of your custom data objects.

```
C#
using Newtonsoft.Json;

public class MyCustomDataType : BaseData
{
    public string Date;
    public decimal Open = 0, High = 0, Low = 0, Close = 0;
    public override DateTime EndTime => Time.AddDays(1);

    public override SubscriptionDataSource GetSource(
        SubscriptionDataConfig config,
        DateTime date,
        bool isLiveMode)
    {
        if (!isLive)
        {
            return new SubscriptionDataSource("<customDataKey>", SubscriptionTransportMedium.ObjectStore,
FileFormat.UnfoldingCollection);
        }
        return new
SubscriptionDataSource("https://raw.githubusercontent.com/QuantConnect/Documentation/master/Resources/datasets/custom-
data/unfolding-collection-example.json", SubscriptionTransportMedium.RemoteFile, FileFormat.UnfoldingCollection);
    }

    public override BaseData Reader(
        SubscriptionDataConfig config,
        string line,
        DateTime date,
        bool isLiveMode)
    {
        if (string.IsNullOrWhiteSpace(line.Trim()))
        {
            return null;
        }

        var objects = JsonConvert.DeserializeObject<List<MyCustomDataType>>(line).Select(index =>
        {
            index.Symbol = config.Symbol;
            index.Time = DateTime.Parse(x.Date, CultureInfo.InvariantCulture);
            index.Value = x.Close;
            return index;
        }).ToList();

        return new BaseDataCollection(objects.Last().EndTime, config.Symbol, objects);
    }
}
```

Create Subscriptions

To create a subscription for the custom type, in the `Initialize` method, call the `AddData` method. The `AddData` method returns a `Security` object, which contains a `Symbol` . Save a reference to the `Symbol` so you can use it in `OnData` to access the security data in the `Slice` .

```
C#
public class MyAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    public override void Initialize()
    {
        _symbol = AddData<MyCustomDataType>("MyCustomDataType", Resolution.Daily).Symbol;
    }
}
```

The `AddData` method creates a subscription for a single custom data asset and adds it to your user-defined universe.

Receive Custom Data

As your data reader reads your custom data file, LEAN adds the data points in the `Slice` it passes to your algorithm's `OnData` method. To collect the custom data, use the `Symbol` or name of your custom data subscription. You can access the `Value` and custom properties of your custom data class from the `Slice` . To access the custom properties, use the custom attribute .

```C#
public class MyAlgorithm : QCAlgorithm
{
    public override void OnData(Slice slice)
    {
        if (slice.ContainsKey(_symbol))
        {
            var customData = slice[_symbol];
            var close = customData.Close;
        }
    }

    // You can also get the data directly with OnData(<dataClass>) method
    public void OnData(MyCustomDataType customData)
    {
        var close = customData.Close;
    }
}
```

Demonstration Algorithms

CustomDataBitcoinAlgorithm.cs C#

Streaming Data

Custom Universes

Custom Universes

Key Concepts

Introduction

A custom universe lets you select a basket of assets from a custom dataset.

Initialize Universes

To add a custom universe to your algorithm, in the `Initialize` method, pass your universe type and a selector function to the `AddUniverse` method.

```
C#
AddUniverse<MyCustomUniverseDataClass>(SelectorFunction);
```

Receive Custom Data

The universe selector function receives a list of your custom objects and must return a list of `Symbol` objects. In the selector function definition, you can use any of the properties of your custom data type. The `Symbol` objects that you return from the selector function set the constituents of the universe.

```
C#
public class MyCustomUniverseAlgorithm : QCAlgorithm
{
 private IEnumerable<Symbol> SelectorFunction(IEnumerable<BaseData> data)
 {
        return (from singleStockData in data.OfType<MyCustomUniverseDataClass>()
               where singleStockData.CustomAttribute1 > 0
               orderby singleStockData.CustomAttribute2 descending
               select singleStockData.Symbol).Take(5);
    }
}
```

Historical Data

To get historical data for a custom data universe , call the `History` method with the `Universe` object. For more information about historical data, see Universes .

Custom Universes

CSV Format Example

Introduction

This page explains how to import custom data for universe selection sourced in CSV format.

Data Format

You must create a file with data in CSV format. Ensure the data in the file is in chronological order.

```
20170704,SPY,QQQ,FB,AAPL,IWM
20170706,QQQ,AAPL,IWM,FB,GOOGL
20170707,IWM,AAPL,FB,BAC,GOOGL
...
20170729,SPY,QQQ,FB,AAPL,IWM
20170801,QQQ,FB,AAPL,IWM,GOOGL
20170802,QQQ,IWM,FB,BAC,GOOGL
```

Define Custom Types

To define a custom data type, inherit the `BaseData` class and override the GetSource and Reader methods.

```
C#
public class StockDataSource : BaseData
{
    public List<Symbol> Symbols { get; set; } = [];
    public override DateTime EndTime => Time.AddDays(1);
    public string Line { get; set; }

    public override SubscriptionDataSource GetSource(SubscriptionDataConfig config, DateTime date, bool isLiveMode)
    {
        if (!isLiveMode)
        {
            return new SubscriptionDataSource("csv-universe-example.csv", SubscriptionTransportMedium.ObjectStore,
FileFormat.Csv);
        }
        return new
SubscriptionDataSource("https://raw.githubusercontent.com/QuantConnect/Documentation/master/Resources/datasets/custom-
data/csv-universe-example.csv", SubscriptionTransportMedium.RemoteFile, FileFormat.Csv);
    }

    public override BaseData Reader(SubscriptionDataConfig config, string line, DateTime date, bool isLiveMode)
    {
        var csv = line.Split(',');
        var stocks = new StockDataSource { Symbol = config.Symbol, Line = line };

        try
        {
            stocks.Time = isLiveMode
                ? DateTime.UtcNow.ConvertFromUtc(config.DataTimeZone)
                : DateTime.ParseExact(csv[0], "yyyyMMdd", null);

            stocks.Symbols.AddRange(csv.Skip(1).Select(ticker =>
            {
                // The tickers are point-in-time. We generate its security identifier for a given date
                // Then we create a Symbol where the Value is the ticker
                var sid = SecurityIdentifier.GenerateEquity(ticker, Market.USA, mappingResolveDate: stocks.Time);
                return new Symbol(sid, ticker);
            }));
        }
        catch { }
        return stocks;
}
```

Initialize Universe

To perform a universe selection with custom data, in the `Initialize` method, call the `AddUniverse` method.

```
C#
public class MyAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        AddUniverse<StockDataSource>(FilterFunction);
    }
}
```

Receive Custom Data

As your data reader reads your custom data file, LEAN adds the data points into a `IEnumerable<StockDataSource>` object it passes to your algorithm's filter function. Your filter function needs to return a list of `Symbol` or `string` object. LEAN automatically subscribes to these new assets and adds them to your algorithm.

```
C#
public class MyAlgorithm : QCAlgorithm
{
    private IEnumerable<Symbol> FilterFunction(IEnumerable<BaseData> data)
    {
        var stockDataSource = data.OfType<StockDataSource>();
        return stockDataSource.SelectMany(x => x.Symbols);
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        Debug(changes.ToString());
    }
}
```

Demonstration Algorithms

DropboxBaseDataUniverseSelectionAlgorithm.cs C#

Custom Universes

JSON Format Example

Introduction

This page explains how to import custom data for universe selection sourced in JSON format.

Data Format

You must create a file with data in JSON format. Ensure the data in the file is in chronological order.

```
[
    {
      "Date": "20170704",
      "Symbols": ["SPY", "QQQ", "FB", "AAPL", "IWM"]
    },
    {
      "Date": "20170706",
      "Symbols": ["QQQ", "AAPL", "IWM", "FB", "GOOGL"]
    },
    ...
    {
      "Date": "20170801",
      "Symbols": ["QQQ", "FB", "AAPL", "IWM", "GOOGL"]
    },
    {
      "Date": "20170802",
      "Symbols": ["QQQ", "IWM", "FB", "BAC", "GOOGL"]
    }
]
```

Define Custom Types

To define a custom data type, inherit the `BaseData` class and override the GetSource and Reader methods.

If you need to create multiple objects in your `Reader` method from a single `line` , follow these steps:

1. In the `GetSource` method, pass `FileFormat.UnfoldingCollection` as the third argument to the `SubscriptionDataSource` constructor.
2. In the `Reader` method, order the objects by their timestamp and then return a `BaseDataCollection(endTime, config.Symbol, objects)` where `objects` is a list of your custom data objects.

```C#
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;

public class StockDataSource : BaseData
{
    public List<Symbol> Symbols { get; set; } = [];
    public override DateTime EndTime => Time.AddDays(1);
    public string Line { get; set; }

    public override SubscriptionDataSource GetSource(SubscriptionDataConfig config, DateTime date, bool isLiveMode)
    {
        if (!isLiveMode)
        {
             return new SubscriptionDataSource("universe-example.json", SubscriptionTransportMedium.ObjectStore,
FileFormat.UnfoldingCollection);
        }
        return new
SubscriptionDataSource("https://raw.githubusercontent.com/QuantConnect/Documentation/master/Resources/datasets/custom-
data/universe-example.json", SubscriptionTransportMedium.RemoteFile, FileFormat.UnfoldingCollection);
    }

    public override BaseData Reader(SubscriptionDataConfig config, string line, DateTime date, bool isLiveMode)
    {
        List<StockDataSource> objects = [];
        foreach(var obj in JsonConvert.DeserializeObject(line) as JArray)
        {
            var stocks = new StockDataSource()
            {
                Time = isLiveMode
                    ? DateTime.UtcNow.ConvertFromUtc(config.DataTimeZone)
                    : DateTime.ParseExact(obj["Date"].Value<string>(), "yyyyMMdd", null),
                Line = JsonConvert.SerializeObject(obj)
            };

            stocks.Symbols.AddRange(obj["Symbols"].Values<string>().Select(ticker =>
            {
                // The tickers are point-in-time. We generate its security identifier for a given date
                // Then we create a Symbol where the Value is the ticker
                var sid = SecurityIdentifier.GenerateEquity(ticker, Market.USA, mappingResolveDate: stocks.Time);
                return new Symbol(sid, ticker);
            }));

            objects.Add(stocks);
        }

        return new BaseDataCollection(objects.Last().EndTime, config.Symbol, objects);
    }
}
```

Initialize Universe

To perform a universe selection with custom data, in the `Initialize` method, call the `AddUniverse` method.

```C#
public class MyAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        AddUniverse<StockDataSource>(FilterFunction);
    }
}
```

Receive Custom Data

As your data reader reads your custom data file, LEAN adds the data points into a `IEnumerable<StockDataSource>` object it passes to your algorithm's filter function. Your filter function needs to return a list of `Symbol` or `string` object. LEAN automatically subscribes to these new assets and adds them to your algorithm.

```
C#
public class MyAlgorithm : QCAlgorithm
{
    private IEnumerable<Symbol> FilterFunction(IEnumerable<BaseData> data)
    {
        var stockDataSource = data.OfType<StockDataSource>();
        return stockDataSource.SelectMany(x => x.Symbols);
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        Debug(changes.ToString());
    }
}
```

Demonstration Algorithms

DropboxBaseDataUniverseSelectionAlgorithm.cs C#

Importing Data

Bulk Downloads

Introduction

There are two techniques to import data into your algorithm. You can either manually import the entire file or stream the file line-by-line into your algorithm's `OnData` event. This page explores importing an entire file for manual use.

Instead of downloading the file from a remote file provider, you can upload the file to the Object Store (with the Algorithm Lab or with the CLI) for faster execution.

Recommended Use Cases

The batch import technique is outside of the LEAN's awareness or control, so it can't enforce good practices. However, the batch import technique is good for the loading the following datasets:

- Loading data into the Object Store
- Trained AI Models
- Well-defined historical price datasets
- Parameters and setting imports such as `Symbol` lists

Download Files

The `Download` method downloads the content served from a local file or URL and then returns it as a string.

Basic Usage

```C#
var file = Download("<filePathOrURL>");
```

Download Method Arguments

The `Download` method can accept header settings, a username, and a password for authentication.

Argument	Data Type	Description	Default Value
`address`	`string`	A string containing the URI to download	
`headers`	`IEnumerable<KeyValuePair<string, string>>`	Defines header values to add to the request	`Enumerable.Empty<KeyValuePair<string, string>>()`
`userName`	`string`	The user name associated with the credentials	`null`
`password`	`string`	The password for the user name associated with the credentials	`null`

Download Request Headers

```
var headers = new Dictionary { { "1", "1" } };
Download(address, headers);
Download(address, headers, userName, password);
```

Save Files

When you download a remote file, save it into the Object Store so that you don't have to download the file again. If you need to import the file multiple times, it's faster to import it from the Object Store rather than repeatedly downloading the file from the remote file provider.

Transport Binary Data

Follow these steps to transport binary files:

1. Add the following imports to your local program:
2. Serialize your object.
3. Save the string representation of your object to one of the supported sources .
4. Download the remote file into your project.
5. Restore the object.

Examples

The following examples demonstrate common practices for bulk downloading data.

Example 1: Download Machine Learning Model

The following algorithm makes use of a Scikit-Learn machine learning model to predict SPY price changes and place orders according to the prediction. To obtain the model, we either retrieve it from the Object Store if there exists any or download it from a Dropbox link using the `download` method.

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
DropboxUniverseSelectionAlgorithm.cs Python

Consolidating Data

Consolidating Data

Getting Started

Introduction

Consolidating data allows you to create bars of any length from smaller bars. Consolidation is commonly used to combine one-minute price bars into longer bars such as 10-20 minute bars. Consolidated bars are helpful because price movement over a longer period can sometimes contain less noise and bars of exotic length are less researched than standard bars, so they may present more opportunities to capture alpha.

To consolidate data, create a `Consolidator` object and register it for data. The built-in consolidators make it easy to create consolidated bars without introducing bugs. In the following sections, we will introduce the different types of consolidators and show you how to shape data into any form.

Basic Usage

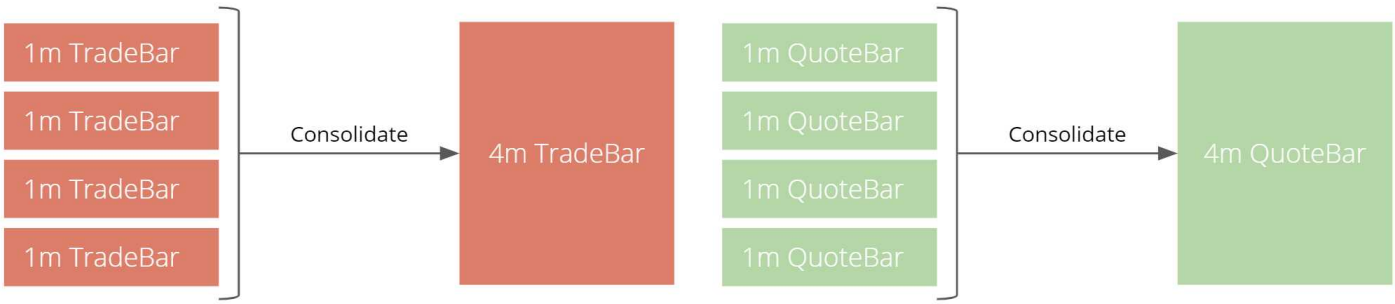
The following code snippet demonstrates a simple consolidator that aggregates 1-minute bars into 10-minutes bars:

```
C#
```

```
// Add some class members
private Symbol _symbol;
private TradeBarConsolidator _consolidator;

// In Initialize, subscribe to a security, create the consolidator, and register it for automatic updates
_symbol = AddEquity("SPY", Resolution.Minute).Symbol;
_consolidator = Consolidate(_symbol, TimeSpan.FromMinutes(10), ConsolidationHandler);

// Define the consolidation handler
void ConsolidationHandler(TradeBar consolidatedBar)
{

}
```

Data Shapes and Sizes

Consolidators usually produce output data that is the same format as the input data. They aggregate small `TradeBar` objects into a large `TradeBar` , small `QuoteBar` objects into a large `QuoteBar` , and `Tick` objects into either a `TradeBar` or `QuoteBar` .



Many asset classes in QuantConnect have data for both trades and quotes. By default, consolidators aggregate data into

`TradeBar` objects. Forex data is the only exception, which consolidators aggregate into `QuoteBar` objects since Forex data doesn't have `TradeBar` objects.

Consolidator Types

There are many different types of consolidators you can create. The process to create and update the consolidator depends on the input data format, output data format, and the consolidation technique.

Time Period Consolidators

Time period consolidators aggregate data based on a period of time like a number of seconds, minutes, or days. If a time period consolidator aggregates data over multiple days, the multi-day aggregation cycle starts and repeats based on the time stamp of the first data point you use to update the consolidator.

The following table shows which consolidator type to use based on the data format of the input and output:

Input	Output	Class Type
`TradeBar`	`TradeBar`	TradeBarConsolidator
`QuoteBar`	`QuoteBar`	QuoteBarConsolidator
`Tick`	`TradeBar`	TickConsolidator
`Tick`	`QuoteBar`	TickQuoteBarConsolidator

Calendar Consolidators

Calendar consolidators aggregate data based on specific periods of time like a weekly basis, quarterly basis, or any schedule with specific start and end times. In contrast to time period consolidators, the bars that calendar consolidators produce always end on the same schedule, regardless of the first data point the consolidator receives.

The following table shows which consolidator type to use based on the data format of the input and output:

Input	Output	Class Type
`TradeBar`	`TradeBar`	TradeBarConsolidator
`QuoteBar`	`QuoteBar`	QuoteBarConsolidator
`Tick`	`TradeBar`	TickConsolidator
`Tick`	`QuoteBar`	TickQuoteBarConsolidator

Count Consolidators

Count consolidators aggregate data based on a number of bars or ticks. This type of consolidator aggregates n samples together, regardless of the time period the samples cover. Managing count consolidators is similiar to managing time period consolidators , except you create count consolidators with an integer argument instead of a time-based argument.

The following table shows which consolidator type to use based on the data format of the input and output:

Input	Output	Class Type
`TradeBar`	`TradeBar`	TradeBarConsolidator
`QuoteBar`	`QuoteBar`	QuoteBarConsolidator
`Tick`	`TradeBar`	TickConsolidator
`Tick`	`QuoteBar`	TickQuoteBarConsolidator

Mixed-Mode Consolidators

Mixed-mode consolidators are a combination of count consolidators and time period consolidators. This type of consolidator aggregates n samples together or aggregates samples over a specific time period, whichever happens first.

The following table shows which consolidator type to use based on the data format of the input and output:

Input	Output	Class Type
`TradeBar`	`TradeBar`	TradeBarConsolidator
`QuoteBar`	`QuoteBar`	QuoteBarConsolidator
`Tick`	`TradeBar`	TickConsolidator
`Tick`	`QuoteBar`	TickQuoteBarConsolidator

Renko Consolidators

Most Renko consolidators aggregate bars based on a fixed price movement. The `RenkoConsolidator` produces Renko bars by their traditional definition. In the case of a $1 bar size, the `RenkoConsolidator` produces bars that have a body spanning 1.*Theopeningpriceofthefirstbarissettotheclosest*1 multiple of the first trade. When the price moves by at least 1, *thefirstbarcloses*. *Ifabarisarisingbar*, *thefollowingbarcloseswhenthepricemoves*1 above the closing price of the previous bar or 1*belowtheopeningpriceofthepreviousbar*. *Ifabarisafallingbar*, *thefollowingbarcloseswhenthepricemoves*1 below the closing price of the previous bar or $1 above the opening price of the previous bar. If the price jumps multiple dollars in a single tick, the `RenkoConsolidator` produces multiple $1 bars in a single time step.

The following table shows which consolidator type to use based on the data format of the input and output:

Input	Output	Class Type
`TradeBar`	`TradeBar`	RenkoConsolidator
`QuoteBar`	`QuoteBar`	RenkoConsolidator
`Tick`	`TradeBar`	RenkoConsolidator
`Tick`	`QuoteBar`	RenkoConsolidator

Classic Renko Consolidators

Most Renko consolidators aggregate bars based on a fixed price movement. The `ClassicRenkoConsolidator` produces a different type of Renko bars than the `RenkoConsolidator` . A `ClassicRenkoConsolidator` with a bar size of

1*producesanewbarthatspans*1 every time an asset closes $1 away from the close of the previous bar. If the price jumps multiple dollars in a single tick, the `ClassicRenkoConsolidator` only produces one bar per time step where the open of each bar matches the close of the previous bar.

The following table shows which consolidator type to use based on the data format of the input and output:

Input	Output	Class Type
`TradeBar`	`TradeBar`	ClassicRenkoConsolidator
`QuoteBar`	`QuoteBar`	ClassicRenkoConsolidator
`Tick`	`TradeBar`	ClassicRenkoConsolidator
`Tick`	`QuoteBar`	ClassicRenkoConsolidator

Volume Renko Consolidators

Volume Renko consolidators aggregate bars based on a fixed trading volume. These types of bars are commonly known as volume bars. A `VolumeRenkoConsolidator` with a bar size of 10,000 produces a new bar every time 10,000 units of the security trades in the market. If the trading volume of a single time step exceeds two step sizes (i.e. >20,000), the `VolumeRenkoConsolidator` produces multiple bars in a single time step.

The following table shows which consolidator type to use based on the data format of the input and output:

Input	Output	Class Type
`TradeBar`	`TradeBar`	VolumeRenkoConsolidator
`Tick`	`TradeBar`	VolumeRenkoConsolidator

Range Consolidators

Most Range consolidators aggregate bars based on a fixed price movement. It is very much like a Renko Bar, but instead of consolidating a fix movement from the opening price, a Range Bar would fix the range of the whole bar, i.e. High - Low. A new bar is formed when the range of the whole bar reached a preset value, thus it completely neglects the time effect and focuses on the variance/information carried by the price movement.

The following table shows which consolidator type to use based on the data format of the input and output:

Input	Output	Class Type
`TradeBar`	`TradeBar`	RangeConsolidator
`QuoteBar`	`QuoteBar`	RangeConsolidator
`Tick`	`TradeBar`	RangeConsolidator
`Tick`	`QuoteBar`	RangeConsolidator

Sequential Consolidators

Sequential consolidators wire two internal consolidators together such that the output of the first consolidator is the input to the

second consolidator and the output of the second consolidator is the output of the sequential consolidator.

For more information about sequential consolidators, see Combining Consolidators .

Execution Sequence

The algorithm manager calls events in the following order:

1. Scheduled Events
2. Consolidation event handlers
3. `OnData` event handler

This event flow is important to note. For instance, if your consolidation handlers or `OnData` event handler appends data to a `RollingWindow` and you use that `RollingWindow` in your Scheduled Event, when the Scheduled Event executes, the `RollingWindow` won't contain the most recent data.

The consolidators are called in the order that they are updated. If you register them for automatic updates, they are updated in the order that you register them.

Plot Consolidated Data

To plot consolidated bars, call the `Plot` method with a `TradeBar` argument in the consolidation handler.

```
C#

// In Initialize
var equity = AddEquity("SPY");
Consolidate(equity.Symbol, TimeSpan.FromMinutes(10), ConsolidationHandler);

// Define the consolidation handler
void ConsolidationHandler(TradeBar consolidatedBar)
{
    Plot("<chartName>", "<seriesName>", consolidatedBar)
}
```

Reset Consolidators

To reset a consolidator, call its `Reset` method.

```
C#

_consolidator.Reset();
```

If you are live trading Equities or backtesting Equities without the adjusted data normalization mode , reset your consolidators when splits and dividends occur. When a split or dividend occurs while the consolidator is in the process of building a bar, the open, high, and low may reflect prices from before the split or dividend. To avoid issues, call the consolidator's `Reset` method and then warm it up with `ScaledRaw` data from a history request .

```
C#
public override void OnData(Slice data)
{
    // When a split or dividend occurs...
    if ((data.Splits.ContainsKey(_symbol) && data.Splits[_symbol].Type == SplitType.SplitOccurred) ||
        data.Dividends.ContainsKey(_symbol))
    {
        // If the consolidator is working on a bar...
        if (_consolidator.WorkingData != null)
        {
            // Get adjusted prices for the time period of the working bar.
            var history = History<TradeBar>(_symbol, _consolidator.WorkingData.Time, Time, dataNormalizationMode:
DataNormalizationMode.ScaledRaw);
            // Reset the consolidator.
            _consolidator.Reset();
            // Warm-up the consolidator with the adjusted price data.
            foreach (var bar in history)
            {
                _consolidator.Update(bar);
            }
        }
    }
}
```

Examples

Demonstration Algorithms
BasicTemplateFuturesConsolidationAlgorithm.cs C# BasicTemplateOptionsConsolidationAlgorithm.cs C#
DataConsolidationAlgorithm.cs C# MultipleSymbolConsolidationAlgorithm.cs C#
Video Tutorials
Quick Start Lesson 4: How do I use Consolidators? C#

Consolidating Data

Consolidator Types

Consolidator Types

Time Period Consolidators

Introduction

Time period consolidators aggregate data based on a period of time like a number of seconds, minutes, or days. If a time period consolidator aggregates data over multiple days, the multi-day aggregation cycle starts and repeats based on the time stamp of the first data point you use to update the consolidator.

Consolidate Trade Bars

`TradeBar` consolidators aggregate `TradeBar` objects into `TradeBar` objects of the same size or larger. Follow these steps to create and manage a `TradeBar` consolidator based on a period of time:

1. Create the consolidator.

   To set the time period for the consolidator, you can use either a `TimeSpan` or `Resolution` object. The consolidator time period must be greater than or equal to the resolution of the security subscription. For instance, you can aggregate minute bars into 10-minute bars, but you can't aggregate hour bars into 10-minute bars.

   - `TimeSpan` Periods

     ```
     C#
     _consolidator = new TradeBarConsolidator(TimeSpan.FromDays(1));
     // Aliases:
     // _consolidator = CreateConsolidator(TimeSpan.FromDays(1), typeof(TradeBar));
     // _consolidator = ResolveConsolidator(_symbol, TimeSpan.FromDays(1));
     ```

     The time period is relative to the data time zone , not the algorithm time zone . If you consolidate Crypto data into daily bars, the event handler receives the consolidated bars at midnight 12:00 AM Coordinated Universal Time (UTC), regardless of the algorithm time zone.

     Use the `startTime` parameter of the TradeBarConsolidator type constructor to define a custom start time.

     ```
     C#
     _consolidator = new TradeBarConsolidator(TimeSpan.FromDays(1),
         startTime: new TimeSpan(9, 30, 0));
     ```

   - `Resolution` Periods

     The `Resolution` enumeration has the following members:

```
_consolidator = TradeBarConsolidator.FromResolution(Resolution.Daily);
// Alias:
// _consolidator = ResolveConsolidator(_symbol, Resolution.Daily);
```

If the security subscription in your algorithm provides `TradeBar` and `QuoteBar` data, `ResolveConsolidator` returns a `TradeBarConsolidator` .

If you consolidate on an hourly basis, the consolidator ends at the top of the hour, not every hour after the market open. For US Equities, that's 10 AM Eastern Time (ET), not 10:30 AM ET. To change this behavior, set the `startTime` parameter of the TradeBarConsolidator type constructor to `TimeSpan(9, 30, 0)`

2. Add an event handler to the consolidator.

```
_consolidator.DataConsolidated += ConsolidationHandler;
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

```
void ConsolidationHandler(object sender, TradeBar consolidatedBar)
{

}
```

The consolidation event handler receives bars when the consolidated bar closes based on the data time zone . If you subscribe to minute resolution data for Bitcoin and create an hourly consolidator, you receive consolidated bars at the top of each hour. However, if you subscribe to minute resolution data for the regular trading hours of US Equities and create a daily consolidator, you receive consolidated bars at market close or midnight, depending on your `DailyPreciseEndTime` setting .

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  SubscriptionManager.AddConsolidator(_symbol, _consolidator);
  ```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To

manually update a consolidator, call its `Update` method with a `TradeBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var tradeBar = slice.Bars[_symbol];
    _consolidator.Update(tradeBar);
}

// Example 2: Update the consolidator with data from a history request
var history = History<TradeBar>(_symbol, 30, Resolution.Minute);
foreach (var tradeBar in history)
{
    _consolidator.Update(tradeBar);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

You can also use the `Consolidate` helper method to create period consolidators and register them for automatic updates. With just one line of code, you can create data in any time period based on a `TimeSpan` or `Resolution` object:

- `TimeSpan` Periods

  ```
  _consolidator = Consolidate(_symbol, TimeSpan.FromDays(1), ConsolidationHandler);
  ```

- `Resolution` Periods

  ```
  _consolidator = Consolidate(_symbol, Resolution.Daily, ConsolidationHandler);
  ```

If you use the `Consolidate` helper method, the consolidation handler doesn't receive an `object` argument.

```
void ConsolidationHandler(TradeBar consolidatedBar)
{

}
```

Consolidate Quote Bars

`QuoteBar` consolidators aggregate `QuoteBar` objects into `QuoteBar` objects of the same size or larger. Follow these steps to create and manage a `QuoteBar` consolidator based on a period of time:

1. Create the consolidator.

   To set the time period for the consolidator, you can use either a `TimeSpan` or `Resolution` object. The consolidator time

period must be greater than or equal to the resolution of the security subscription. For instance, you can aggregate minute bars into 10-minute bars, but you can't aggregate hour bars into 10-minute bars.

- `TimeSpan` Periods

```
C#
_consolidator = new QuoteBarConsolidator(TimeSpan.FromDays(1));
// Aliases:
// _consolidator = CreateConsolidator(TimeSpan.FromDays(1), typeof(QuoteBar));
// _consolidator = ResolveConsolidator(_symbol, TimeSpan.FromDays(1));
```

The time period is relative to the data time zone , not the algorithm time zone . If you consolidate Crypto data into daily bars, the event handler receives the consolidated bars at midnight 12:00 AM Coordinated Universal Time (UTC), regardless of the algorithm time zone.

Use the `startTime` parameter of the QuoteBarConsolidator type constructor to define a custom start time.

```
C#
_consolidator = new QuoteBarConsolidator(TimeSpan.FromDays(1),
    startTime: new TimeSpan(9, 30, 0));
```

- `Resolution` Periods

The `Resolution` enumeration has the following members:

```
C#
_consolidator = ResolveConsolidator(_symbol, Resolution.Daily);
```

If the security subscription in your algorithm provides `TradeBar` and `QuoteBar` data, `ResolveConsolidator` returns a `TradeBarConsolidator` .

If you consolidate on an hourly basis, the consolidator ends at the top of the hour, not every hour after the market open. For US Equities, that's 10 AM Eastern Time (ET), not 10:30 AM ET. To change this behavior, set the `startTime` parameter of the QuoteBarConsolidator type constructor to `TimeSpan(9, 30, 0)`

2. Add an event handler to the consolidator.

```
C#
_consolidator.DataConsolidated += ConsolidationHandler;
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

```
C#
void ConsolidationHandler(object sender, QuoteBar consolidatedBar)
{

}
```

The consolidation event handler receives bars when the consolidated bar closes based on the data time zone . If you subscribe to minute resolution data for Bitcoin and create an hourly consolidator, you receive consolidated bars at the top of each hour. However, if you subscribe to minute resolution data for the regular trading hours of US Equities and create a daily consolidator, you receive consolidated bars at market close or midnight, depending on your `DailyPreciseEndTime` setting .

4. Update the consolidator.

   You can automatically or manually update the consolidator.

   - Automatic Updates

     To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

     ```
     C#

     SubscriptionManager.AddConsolidator(_symbol, _consolidator);
     ```

   - Manual Updates

     Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `QuoteBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

     ```
     C#

     // Example 1: Update the consolidator with data from the Slice object
     public override void OnData(Slice slice)
     {
         var quoteBar = slice.QuoteBars[_symbol];
         _consolidator.Update(quoteBar);
     }

     // Example 2: Update the consolidator with data from a history request
     var history = History<QuoteBar>(_symbol, 30, Resolution.Minute);
     foreach (var quoteBar in history)
     {
         _consolidator.Update(quoteBar);
     }
     ```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   C#

   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

You can also use the `Consolidate` helper method to create period consolidators and register them for automatic updates. With just one line of code, you can create data in any time period based on a `TimeSpan` or `Resolution` object:

- `TimeSpan` Periods

```C#
_consolidator = Consolidate<QuoteBar>(_symbol, TimeSpan.FromDays(1), TickType.Quote, ConsolidationHandler);
```

- `Resolution` Periods

```C#
_consolidator = Consolidate<QuoteBar>(_symbol, Resolution.Daily, TickType.Quote, ConsolidationHandler);
```

If you use the `Consolidate` helper method, the consolidation handler doesn't receive an `object` argument.

```C#
void ConsolidationHandler(QuoteBar consolidatedBar)
{

}
```

Consolidate Trade Ticks

`Tick` consolidators aggregate `Tick` objects into `TradeBar` objects. Follow these steps to create and manage a `Tick` consolidator based on a period of time:

1. Create the consolidator.

   To set the time period for the consolidator, you can use either a `TimeSpan` or `Resolution` object. The consolidator time period must be greater than or equal to the resolution of the security subscription. For instance, you can aggregate minute bars into 10-minute bars, but you can't aggregate hour bars into 10-minute bars.

   - `TimeSpan` Periods

     ```C#
     _consolidator = new TickConsolidator(TimeSpan.FromMilliseconds(100));
     // Aliases:
     // _consolidator = CreateConsolidator(TimeSpan.FromMilliseconds(100), typeof(Tick));
     // _consolidator = ResolveConsolidator(_symbol, TimeSpan.FromMilliseconds(100));
     ```

     The time period is relative to the data time zone , not the algorithm time zone . If you consolidate Crypto data into daily bars, the event handler receives the consolidated bars at midnight 12:00 AM Coordinated Universal Time (UTC), regardless of the algorithm time zone.

     Use the `startTime` parameter of the TickConsolidator type constructor to define a custom start time.

     ```C#
     _consolidator = new TickConsolidator(TimeSpan.FromMilliseconds(100),
         startTime: new TimeSpan(9, 30, 0));
     ```

   - `Resolution` Periods

     The `Resolution` enumeration has the following members:

     ```C#
     _consolidator = ResolveConsolidator(_symbol, Resolution.Second);
     ```

   If you consolidate on an hourly basis, the consolidator ends at the top of the hour, not every hour after the market open.

For US Equities, that's 10 AM Eastern Time (ET), not 10:30 AM ET. To change this behavior, set the `startTime` parameter of the TickConsolidator type constructor to `TimeSpan(9, 30, 0)`

2. Add an event handler to the consolidator.

```
C#
_consolidator.DataConsolidated += ConsolidationHandler;
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

```
C#
void ConsolidationHandler(object sender, TradeBar consolidatedBar)
{

}
```

The consolidation event handler receives bars when the consolidated bar closes based on the data time zone . If you subscribe to tick resolution data for Bitcoin and create an hourly consolidator, you receive consolidated bars at the top of each hour. However, if you subscribe to tick resolution data for the regular trading hours of US Equities and create a daily consolidator, you receive consolidated bars at market close or midnight, depending on your `DailyPreciseEndTime` setting .

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
C#
SubscriptionManager.AddConsolidator(_symbol, _consolidator);
```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```C#
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var ticks = slice.Ticks[_symbol];
    foreach (var tick in ticks)
    {
     _consolidator.Update(tick);
    }
}

// Example 2: Update the consolidator with data from a history request
var ticks = History<Tick>(_symbol, TimeSpan.FromMinutes(3), Resolution.Tick);
foreach (var tick in ticks)
{
    _consolidator.Update(tick);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```C#
   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

You can also use the `Consolidate` helper method to create period consolidators and register them for automatic updates. With just one line of code, you can create data in any time period based on a `TimeSpan` or `Resolution` object:

- `TimeSpan` Periods

  ```C#
  _consolidator = Consolidate(_symbol, TimeSpan.FromMilliseconds(100), ConsolidationHandler);
  ```

- `Resolution` Periods

  ```C#
  _consolidator = Consolidate(_symbol, Resolution.Second, ConsolidationHandler);
  ```

If you use the `Consolidate` helper method, the consolidation handler doesn't receive an `object` argument.

```C#
void ConsolidationHandler(TradeBar consolidatedBar)
{

}
```

Consolidate Quote Ticks

`Tick` quote bar consolidators aggregate `Tick` objects that represent quotes into `QuoteBar` objects. Follow these steps to create and manage a `Tick` quote bar consolidator based on a period of time:

1. Create the consolidator.

   To set the time period for the consolidator, you can use either a `TimeSpan` or `Resolution` object. The consolidator time period must be greater than or equal to the resolution of the security subscription. For instance, you can aggregate minute

bars into 10-minute bars, but you can't aggregate hour bars into 10-minute bars.

- `TimeSpan` Periods

  ```
  C#

  _consolidator = new TickQuoteBarConsolidator(TimeSpan.FromMilliseconds(100));
  // Aliases:
  // _consolidator = CreateConsolidator(TimeSpan.FromMilliseconds(100), typeof(Tick), TickType.Quote);
  // _consolidator = ResolveConsolidator(_symbol, TimeSpan.FromMilliseconds(100), typeof(QuoteBar));
  ```

  The time period is relative to the data time zone , not the algorithm time zone . If you consolidate Crypto data into daily bars, the event handler receives the consolidated bars at midnight 12:00 AM Coordinated Universal Time (UTC), regardless of the algorithm time zone.

  Use the `startTime` parameter of the TickQuoteBarConsolidator type constructor to define a custom start time.

  ```
  C#

  _consolidator = new TickQuoteBarConsolidator(TimeSpan.FromMilliseconds(100),
      startTime: new TimeSpan(9, 30, 0));
  ```

- `Resolution` Periods

  The `Resolution` enumeration has the following members:

  ```
  C#

  _consolidator = ResolveConsolidator(_symbol, Resolution.Second);
  ```

If you consolidate on an hourly basis, the consolidator ends at the top of the hour, not every hour after the market open. For US Equities, that's 10 AM Eastern Time (ET), not 10:30 AM ET. To change this behavior, set the `startTime` parameter of the TickQuoteBarConsolidator type constructor to `TimeSpan(9, 30, 0)`

2. Add an event handler to the consolidator.

   ```
   C#

   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

   ```
   C#

   void ConsolidationHandler(object sender, QuoteBar consolidatedBar)
   {

   }
   ```

   The consolidation event handler receives bars when the consolidated bar closes based on the data time zone . If you subscribe to tick resolution data for Bitcoin and create an hourly consolidator, you receive consolidated bars at the top of each hour. However, if you subscribe to tick resolution data for the regular trading hours of US Equities and create a daily

consolidator, you receive consolidated bars at market close or midnight, depending on your `DailyPreciseEndTime` setting .

4. Update the consolidator.

   You can automatically or manually update the consolidator.

   - Automatic Updates

     To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

     ```
     C#

     SubscriptionManager.AddConsolidator(_symbol, _consolidator);
     ```

   - Manual Updates

     Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

     ```
     C#

     // Example 1: Update the consolidator with data from the Slice object
     public override void OnData(Slice slice)
     {
         var ticks = slice.Ticks[_symbol];
         foreach (var tick in ticks)
         {
          _consolidator.Update(tick);
         }
     }

     // Example 2: Update the consolidator with data from a history request
     var ticks = History<Tick>(_symbol, TimeSpan.FromMinutes(3), Resolution.Tick);
     foreach (var tick in ticks)
     {
         _consolidator.Update(tick);
     }
     ```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   C#

   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

You can also use the `Consolidate` helper method to create period consolidators and register them for automatic updates. With just one line of code, you can create data in any time period based on a `TimeSpan` or `Resolution` object:

- `TimeSpan` Periods

  ```
  C#

  _consolidator = Consolidate<QuoteBar>(_symbol, TimeSpan.FromMilliseconds(100), TickType.Quote,
  ConsolidationHandler);
  ```

- `Resolution` Periods

```
C#
_consolidator = Consolidate<QuoteBar>(_symbol, Resolution.Second, TickType.Quote, ConsolidationHandler);
```

If you use the `Consolidate` helper method, the consolidation handler doesn't receive an `object` argument.

```
C#
void ConsolidationHandler(QuoteBar consolidatedBar)
{

}
```

Consolidate Other Data

`<dataType>` consolidators aggregate various types of data objects into `<dataType>` objects of the same size or larger. If you consolidate custom data or alternative datasets, check the definition of the data class to see its data type. Follow these steps to create and manage a `<dataType>` consolidator based on a period of time:

1. Create the consolidator.

   To set the time period for the consolidator, you can use either a `TimeSpan` or `Resolution` object. The consolidator time period must be greater than or equal to the resolution of the security subscription. For instance, you can aggregate minute bars into 10-minute bars, but you can't aggregate hour bars into 10-minute bars.

   - `TimeSpan` Periods

     ```
     C#
     _consolidator = new DynamicDataConsolidator(TimeSpan.FromDays(1));
     // Aliases:
     // _consolidator = CreateConsolidator(TimeSpan.FromDays(1), typeof(<dataType>));
     // _consolidator = ResolveConsolidator(_symbol, TimeSpan.FromDays(1));
     ```

     The time period is relative to the data time zone , not the algorithm time zone . If you consolidate Crypto data into daily bars, the event handler receives the consolidated bars at midnight 12:00 AM Coordinated Universal Time (UTC), regardless of the algorithm time zone.

     Use the `startTime` parameter of the DynamicDataConsolidator type constructor to define a custom start time.

     ```
     C#
     _consolidator = new DynamicDataConsolidator(TimeSpan.FromDays(1),
         startTime: new TimeSpan(9, 30, 0));
     ```

   - `Resolution` Periods

     The `Resolution` enumeration has the following members:

     ```
     C#
     _consolidator = ResolveConsolidator(_symbol, Resolution.Daily);
     ```

   If you consolidate on an hourly basis, the consolidator ends at the top of the hour, not every hour after the market open. For US Equities, that's 10 AM Eastern Time (ET), not 10:30 AM ET. To change this behavior, set the `startTime` parameter of the DynamicDataConsolidator type constructor to `TimeSpan(9, 30, 0)`

2. Add an event handler to the consolidator.

```C#
_consolidator.DataConsolidated += ConsolidationHandler;
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

```C#
void ConsolidationHandler(object sender, <dataType> consolidatedBar)
{

}
```

The consolidation event handler receives bars when the consolidated bar closes based on the data time zone .

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the data subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```C#
  SubscriptionManager.AddConsolidator(_symbol, _consolidator);
  ```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `<dataType>` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

  ```C#
  // Example 1: Update the consolidator with data from the Slice object
  public override void OnData(Slice slice)
  {
      if slice.ContainsKey(_symbol)
      {
          _consolidator.Update(slice[_symbol]);
      }
  }

  // Example 2: Update the consolidator with data from a history request
  var history = History<<dataType>>(_symbol, 30, Resolution.Daily);
  foreach (var data in history)
  {
      _consolidator.Update(data);
  }
  ```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the

consolidator when the security leaves the universe.

```
SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

You can also use the `Consolidate` helper method to create period consolidators and register them for automatic updates. With just one line of code, you can create data in any time period based on a `TimeSpan` or `Resolution` object:

- `TimeSpan` Periods

```
_consolidator = Consolidate(_symbol, TimeSpan.FromDays(1), ConsolidationHandler);
```

- `Resolution` Periods

```
_consolidator = Consolidate(_symbol, Resolution.Daily, ConsolidationHandler);
```

If you use the `Consolidate` helper method, the consolidation handler doesn't receive an `object` argument.

```
void ConsolidationHandler(<dataType> consolidatedBar)
{

}
```

Reset Consolidators

To reset a consolidator, call its `Reset` method.

```
_consolidator.Reset();
```

If you are live trading Equities or backtesting Equities without the adjusted data normalization mode , reset your consolidators when splits and dividends occur. When a split or dividend occurs while the consolidator is in the process of building a bar, the open, high, and low may reflect prices from before the split or dividend. To avoid issues, call the consolidator's `Reset` method and then warm it up with `ScaledRaw` data from a history request .

```C#
public override void OnData(Slice data)
{
    // When a split or dividend occurs...
    if ((data.Splits.ContainsKey(_symbol) && data.Splits[_symbol].Type == SplitType.SplitOccurred) ||
        data.Dividends.ContainsKey(_symbol))
    {
        // If the consolidator is working on a bar...
        if (_consolidator.WorkingData != null)
        {
            // Get adjusted prices for the time period of the working bar.
            var history = History<TradeBar>(_symbol, _consolidator.WorkingData.Time, Time, dataNormalizationMode:
DataNormalizationMode.ScaledRaw);
            // Reset the consolidator.
            _consolidator.Reset();
            // Warm-up the consolidator with the adjusted price data.
            foreach (var bar in history)
            {
                _consolidator.Update(bar);
            }
        }
    }
}
```

Examples

The following examples demonstrate common practices for implementing time-period consolidators.

Example 1: RSI-MACD Trading

The following algorithm utilizes a 5-minute trade bar consolidator to feed the smoothed trade bar to an RSI indicator and a 15-minute quote bar consolidator to feed the smoothed quote bar to a MACD indicator, generating trade entry signals during an uptrend and oversold conditions while exiting when the trend turns down.

C#

```
public class TimePeriodConsolidatorAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    // Create RSI and MACD indicators for trade signal generation.
    private RelativeStrengthIndex _rsi = new(14);
    private MovingAverageConvergenceDivergence _macd = new(12, 26, 9);

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request SPY data to feed to indicators and trade.
        _spy = AddEquity("SPY").Symbol;

        // Create a 5-minute trade bar consolidator to feed the smoothed bar to the RSI indicator for the price
action entry signal.
        var rsiConsolidator = new TradeBarConsolidator(TimeSpan.FromMinutes(5));
        // Create a 15-minute quote bar consolidator to feed the smoothed bar to the MACD consolidator for trend
entry/exit signal.
        var macdConsolidator = new QuoteBarConsolidator(TimeSpan.FromMinutes(15));

        // Subscribe the consolidators to SPY data to automatically update the indicators.
        RegisterIndicator(_spy, _rsi, rsiConsolidator);
        RegisterIndicator(_spy, _macd, macdConsolidator);

        // Warm up the indicators by updating the consolidators with historical data.
        var tradeBars = History<TradeBar>(_spy, 5 * _rsi.WarmUpPeriod, Resolution.Minute);
        foreach (var bar in tradeBars)
        {
            rsiConsolidator.Update(bar);
        }
        var quoteBars = History<QuoteBar>(_spy, 15 * _macd.WarmUpPeriod, Resolution.Minute);
        foreach (var bar in quoteBars)
        {
            macdConsolidator.Update(bar);
        }
    }

    public override void OnData(Slice slice)
    {
        // Entry rule: To maximize return, the MACD histogram must be positive (positive trend) and the RSI must be
below 30 (oversold).
        if (_rsi < 30 && _macd.Histogram > 0 && !Portfolio[_spy].IsLong)
        {
            SetHoldings(_spy, 0.5m);
        }
        // Exit rule: MACD histogram is below 0 (negative trend)
        else if (_macd.Histogram < 0 && Portfolio[_spy].IsLong)
        {
            Liquidate(_spy);
        }
    }
}
```

Example 2: Create Daily `QuoteBar` Objects That Start at 5 PM

The following algorithm consolidates minute resolution Forex data into daily `QuoteBar` objects that open and close at 5 PM Eastern Time.

C#

```
public class CustomStartTimeConsolidatorExampleAlgorithm : QCAlgorithm
{
    private Forex _pair;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add the USDJPY Forex pair.
        _pair = AddForex("USDJPY");
        // Create an EMA indicator for the pair.
        var ema = new ExponentialMovingAverage(10);
        _pair.Set("Ema", ema);
        // Create a QuoteBar consolidator with a custom start time.
        var consolidator = new QuoteBarConsolidator(TimeSpan.FromDays(1), TimeSpan.FromHours(17));
        // Attach a consolidation handler that will receive the consolidated bars.
        consolidator.DataConsolidated += ConsolidationHandler;
        // Subscribe the consolidator for automatic updates with the prices of the pair.
        SubscriptionManager.AddConsolidator(_pair.Symbol, consolidator);
        // Register the indicator for automatic updates with the consolidated bars.
        RegisterIndicator<IndicatorDataPoint>(_pair.Symbol, ema, consolidator);
        // Warm up the consolidator and indicator.
        var history = History<QuoteBar>(_pair.Symbol, 29000, Resolution.Minute);
        foreach (var bar in history)
        {
            consolidator.Update(bar);
        }
    }

    private void ConsolidationHandler(object sender, QuoteBar consolidatedBar)
    {
        // Wait until the indicator is ready and the algorithm is running.
        var ema = _pair.Get<ExponentialMovingAverage>("Ema");
        if (!ema.IsReady || IsWarmingUp)
        {
            return;
        }
        // Plot the closing price and the EMA.
        Plot(consolidatedBar.Symbol.Value, "Close", consolidatedBar.Close);
        Plot(consolidatedBar.Symbol.Value, "EMA", ema.Current.Value);

        if (!_pair.Holdings.IsLong && consolidatedBar.Close > ema)
        {
            SetHoldings(_pair.Symbol, 1);
        }
        if (consolidatedBar.Close < ema)
        {
            Liquidate();
        }
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
BasicTemplateOptionsConsolidationAlgorithm.cs C# BasicTemplateFuturesConsolidationAlgorithm.cs C#
DataConsolidationAlgorithm.cs C# MultipleSymbolConsolidationAlgorithm.cs C#

Consolidator Types

Calendar Consolidators

Introduction

Calendar consolidators aggregate data based on specific periods of time like a weekly basis, quarterly basis, or any schedule with specific start and end times. In contrast to time period consolidators, the bars that calendar consolidators produce always end on the same schedule, regardless of the first data point the consolidator receives.

Consolidate Trade Bars

`TradeBar` consolidators aggregate `TradeBar` objects into `TradeBar` objects of the same size or larger. Follow these steps to create and manage a `TradeBar` consolidator based on custom start and end periods:

1. Create the consolidator.

   To set the time period for the consolidator, you can use the built-in `CalendarInfo` objects or create your own. The following list describes each technique:

   - Standard Periods

     The following table describes the helper methods that the `Calendar` class provides to create the built-in `CalendarInfo` objects:

Method	Description
`Calendar.Weekly`	Computes the start of week (previous Monday) of the given date/time
`Calendar.Monthly`	Computes the start of month (1st of the current month) of the given date/time
`Calendar.Quarterly`	Computes the start of quarter (1st of the starting month of the current quarter) of the given date/time
`Calendar.Yearly`	Computes the start of year (1st of the current year) of the given date/time

     ```
     C#
     _consolidator = new TradeBarConsolidator(Calendar.Weekly);
     // Alias:
     // _consolidator = CreateConsolidator(Calendar.Weekly, typeof(TradeBar));
     ```

   - Custom Periods

     If you need something more specific than the preceding time periods, define a method to set the start time and period of the consolidated bars. The method should receive a `DateTime` object that's based in the data time zone and should return a `CalendarInfo` object, which contains the start time of the bar in the data time zone and the duration of the consolidation period. The following example demonstrates how to create a custom consolidator for weekly bars:

```C#
_consolidator = new TradeBarConsolidator(datetime => {
    var period = TimeSpan.FromDays(7);

    var timeSpan = new TimeSpan(17, 0, 0);
    var newDateTime = datetime.Date + timeSpan;
    var delta = 1 + (int)newDateTime.DayOfWeek;
    if (delta > 6)
    {
        delta = 0;
    }
    var start = newDateTime.AddDays(-delta);

    return new CalendarInfo(start, period);
});
```

2. Add an event handler to the consolidator.

```C#
_consolidator.DataConsolidated += ConsolidationHandler;
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

```C#
void ConsolidationHandler(object sender, TradeBar consolidatedBar)
{

}
```

If you use a custom consolidation period method, LEAN passes the consolidated bar to the consolidation handler when the consolidation period ends. The `Time` and `EndTime` properties of the consolidated bar reflect the data time zone, but the `Time` property of the algorithm still reflects the algorithm time zone .

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```C#
SubscriptionManager.AddConsolidator(_symbol, _consolidator);
```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `TradeBar` object. You can update the consolidator with

data from the Slice object in the `OnData` method or with data from a history request .

```
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var tradeBar = slice.Bars[_symbol];
    _consolidator.Update(tradeBar);
}

// Example 2: Update the consolidator with data from a history request
var history = History<TradeBar>(_symbol, 30, Resolution.Minute);
foreach (var tradeBar in history)
{
    _consolidator.Update(tradeBar);
}
```

C#

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   C#

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

You can also use the `Consolidate` helper method to create calendar consolidators and register them for automatic updates.

```
_consolidator = Consolidate(_symbol, Calendar.Weekly, ConsolidationHandler);
```

C#

If you use the `Consolidate` helper method, the consolidation handler doesn't receive an `object` argument.

```
void ConsolidationHandler(TradeBar consolidatedBar)
{

}
```

C#

Consolidate Quote Bars

`QuoteBar` consolidators aggregate `QuoteBar` objects into `QuoteBar` objects of the same size or larger. Follow these steps to create and manage a `QuoteBar` consolidator based on custom start and end periods:

1. Create the consolidator.

   To set the time period for the consolidator, you can use the built-in `CalendarInfo` objects or create your own. The following list describes each technique:

   - Standard Periods

     The following table describes the helper methods that the `Calendar` class provides to create the built-in `CalendarInfo` objects:

Method	Description
`Calendar.Weekly`	Computes the start of week (previous Monday) of the given date/time
`Calendar.Monthly`	Computes the start of month (1st of the current month) of the given date/time
`Calendar.Quarterly`	Computes the start of quarter (1st of the starting month of the current quarter) of the given date/time
`Calendar.Yearly`	Computes the start of year (1st of the current year) of the given date/time

```
C#
_consolidator = new QuoteBarConsolidator(Calendar.Weekly);
// Alias:
// _consolidator = CreateConsolidator(Calendar.Weekly, typeof(QuoteBar));
```

- Custom Periods

  If you need something more specific than the preceding time periods, define a method to set the start time and period of the consolidated bars. The method should receive a `DateTime` object that's based in the data time zone and should return a `CalendarInfo` object, which contains the start time of the bar in the data time zone and the duration of the consolidation period. The following example demonstrates how to create a custom consolidator for weekly bars:

```
C#
_consolidator = new QuoteBarConsolidator(datetime => {
    var period = TimeSpan.FromDays(7);

    var timeSpan = new TimeSpan(17, 0, 0);
    var newDateTime = datetime.Date + timeSpan;
    var delta = 1 + (int)newDateTime.DayOfWeek;
    if (delta > 6)
    {
        delta = 0;
    }
    var start = newDateTime.AddDays(-delta);

    return new CalendarInfo(start, period);
});
```

2. Add an event handler to the consolidator.

```
C#
_consolidator.DataConsolidated += ConsolidationHandler;
```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler`, not `ConsolidationHandler()`.

3. Define the consolidation handler.

```
C#
void ConsolidationHandler(object sender, QuoteBar consolidatedBar)
{

}
```

If you use a custom consolidation period method, LEAN passes the consolidated bar to the consolidation handler when the consolidation period ends. The `Time` and `EndTime` properties of the consolidated bar reflect the data time zone, but the `Time` property of the algorithm still reflects the algorithm time zone .

4. Update the consolidator.

   You can automatically or manually update the consolidator.

   - Automatic Updates

     To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

     ```
     C#
     SubscriptionManager.AddConsolidator(_symbol, _consolidator);
     ```

   - Manual Updates

     Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `QuoteBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

     ```
     C#
     // Example 1: Update the consolidator with data from the Slice object
     public override void OnData(Slice slice)
     {
         var quoteBar = slice.QuoteBars[_symbol];
         _consolidator.Update(quoteBar);
     }

     // Example 2: Update the consolidator with data from a history request
     var history = History<QuoteBar>(_symbol, 30, Resolution.Minute);
     foreach (var quoteBar in history)
     {
         _consolidator.Update(quoteBar);
     }
     ```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   C#
   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

You can also use the `Consolidate` helper method to create calendar consolidators and register them for automatic updates.

```
_consolidator = Consolidate<QuoteBar>(_symbol, Calendar.Weekly, TickType.Quote, ConsolidationHandler);
```

If you use the `Consolidate` helper method, the consolidation handler doesn't receive an `object` argument.

```
void ConsolidationHandler(QuoteBar consolidatedBar)
{

}
```

Consolidate Trade Ticks

`Tick` consolidators aggregate `Tick` objects into `TradeBar` objects. Follow these steps to create and manage a `Tick` consolidator based on custom start and end periods:

1. Create the consolidator.

   To set the time period for the consolidator, you can use the built-in `CalendarInfo` objects or create your own. The following list describes each technique:

   - Standard Periods

     The following table describes the helper methods that the `Calendar` class provides to create the built-in `CalendarInfo` objects:

Method	Description
`Calendar.Weekly`	Computes the start of week (previous Monday) of the given date/time
`Calendar.Monthly`	Computes the start of month (1st of the current month) of the given date/time
`Calendar.Quarterly`	Computes the start of quarter (1st of the starting month of the current quarter) of the given date/time
`Calendar.Yearly`	Computes the start of year (1st of the current year) of the given date/time

     ```
     _consolidator = new TickConsolidator(Calendar.Weekly);
     // Alias:
     // _consolidator = CreateConsolidator(Calendar.Weekly, typeof(Tick));
     ```

   - Custom Periods

     If you need something more specific than the preceding time periods, define a method to set the start time and period of the consolidated bars. The method should receive a `DateTime` object that's based in the data time zone and should return a `CalendarInfo` object, which contains the start time of the bar in the data time zone and the duration of the consolidation period. The following example demonstrates how to create a custom consolidator for weekly bars:

```
C#
_consolidator = new TickConsolidator(datetime => {
    var period = TimeSpan.FromDays(7);

    var timeSpan = new TimeSpan(17, 0, 0);
    var newDateTime = datetime.Date + timeSpan;
    var delta = 1 + (int)newDateTime.DayOfWeek;
    if (delta > 6)
    {
        delta = 0;
    }
    var start = newDateTime.AddDays(-delta);

    return new CalendarInfo(start, period);
});
```

2. Add an event handler to the consolidator.

```
C#
_consolidator.DataConsolidated += ConsolidationHandler;
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

```
C#
void ConsolidationHandler(object sender, TradeBar consolidatedBar)
{

}
```

If you use a custom consolidation period method, LEAN passes the consolidated bar to the consolidation handler when the consolidation period ends. The `Time` and `EndTime` properties of the consolidated bar reflect the data time zone, but the `Time` property of the algorithm still reflects the algorithm time zone .

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
C#
SubscriptionManager.AddConsolidator(_symbol, _consolidator);
```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data

from the Slice object in the `OnData` method or with data from a history request .

```
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var ticks = slice.Ticks[_symbol];
    foreach (var tick in ticks)
    {
     _consolidator.Update(tick);
    }
}

// Example 2: Update the consolidator with data from a history request
var ticks = History<Tick>(_symbol, TimeSpan.FromMinutes(3), Resolution.Tick);
foreach (var tick in ticks)
{
    _consolidator.Update(tick);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

You can also use the `Consolidate` helper method to create calendar consolidators and register them for automatic updates.

```
_consolidator = Consolidate(_symbol, Calendar.Weekly, ConsolidationHandler);
```

If you use the `Consolidate` helper method, the consolidation handler doesn't receive an `object` argument.

```
void ConsolidationHandler(TradeBar consolidatedBar)
{

}
```

Consolidate Quote Ticks

`Tick` quote bar consolidators aggregate `Tick` objects that represent quotes into `QuoteBar` objects. Follow these steps to create and manage a `Tick` quote bar consolidator based on custom start and end periods:

1. Create the consolidator.

   To set the time period for the consolidator, you can use the built-in `CalendarInfo` objects or create your own. The following list describes each technique:

   - Standard Periods

     The following table describes the helper methods that the `Calendar` class provides to create the built-in `CalendarInfo` objects:

Method	Description
`Calendar.Weekly`	Computes the start of week (previous Monday) of the given date/time
`Calendar.Monthly`	Computes the start of month (1st of the current month) of the given date/time
`Calendar.Quarterly`	Computes the start of quarter (1st of the starting month of the current quarter) of the given date/time
`Calendar.Yearly`	Computes the start of year (1st of the current year) of the given date/time

```
C#
_consolidator = new TickQuoteBarConsolidator(Calendar.Weekly);
// Alias:
// _consolidator = CreateConsolidator(Calendar.Weekly, typeof(Tick), TickType.Quote);
```

- Custom Periods

  If you need something more specific than the preceding time periods, define a method to set the start time and period of the consolidated bars. The method should receive a `DateTime` object that's based in the data time zone and should return a `CalendarInfo` object, which contains the start time of the bar in the data time zone and the duration of the consolidation period. The following example demonstrates how to create a custom consolidator for weekly bars:

```
C#
_consolidator = new TickQuoteBarConsolidator(datetime => {
    var period = TimeSpan.FromDays(7);

    var timeSpan = new TimeSpan(17, 0, 0);
    var newDateTime = datetime.Date + timeSpan;
    var delta = 1 + (int)newDateTime.DayOfWeek;
    if (delta > 6)
    {
        delta = 0;
    }
    var start = newDateTime.AddDays(-delta);

    return new CalendarInfo(start, period);
});
```

2. Add an event handler to the consolidator.

```
C#
_consolidator.DataConsolidated += ConsolidationHandler;
```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler`, not `ConsolidationHandler()`.

3. Define the consolidation handler.

```
C#
void ConsolidationHandler(object sender, QuoteBar consolidatedBar)
{

}
```

If you use a custom consolidation period method, LEAN passes the consolidated bar to the consolidation handler when the consolidation period ends. The `Time` and `EndTime` properties of the consolidated bar reflect the data time zone, but the `Time` property of the algorithm still reflects the algorithm time zone .

4. Update the consolidator.

   You can automatically or manually update the consolidator.

   - Automatic Updates

     To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

     ```
     C#
     SubscriptionManager.AddConsolidator(_symbol, _consolidator);
     ```

   - Manual Updates

     Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

     ```
     C#
     // Example 1: Update the consolidator with data from the Slice object
     public override void OnData(Slice slice)
     {
         var ticks = slice.Ticks[_symbol];
         foreach (var tick in ticks)
         {
          _consolidator.Update(tick);
         }
     }

     // Example 2: Update the consolidator with data from a history request
     var ticks = History<Tick>(_symbol, TimeSpan.FromMinutes(3), Resolution.Tick);
     foreach (var tick in ticks)
     {
         _consolidator.Update(tick);
     }
     ```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   C#
   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

You can also use the `Consolidate` helper method to create calendar consolidators and register them for automatic updates.

```
C#
_consolidator = Consolidate<QuoteBar>(_symbol, Calendar.Weekly, TickType.Quote, ConsolidationHandler);
```

If you use the `Consolidate` helper method, the consolidation handler doesn't receive an `object` argument.

```
C#
void ConsolidationHandler(QuoteBar consolidatedBar)
{

}
```

Consolidate Other Data

`<dataType>` consolidators aggregate various types of data objects into `<dataType>` objects of the same size or larger. If you consolidate custom data or alternative datasets, check the definition of the data class to see its data type. Follow these steps to create and manage a `<dataType>` consolidator based on custom start and end periods:

1. Create the consolidator.

   To set the time period for the consolidator, you can use the built-in `CalendarInfo` objects or create your own. The following list describes each technique:

   - Standard Periods

     The following table describes the helper methods that the `Calendar` class provides to create the built-in `CalendarInfo` objects:

Method	Description
`Calendar.Weekly`	Computes the start of week (previous Monday) of the given date/time
`Calendar.Monthly`	Computes the start of month (1st of the current month) of the given date/time
`Calendar.Quarterly`	Computes the start of quarter (1st of the starting month of the current quarter) of the given date/time
`Calendar.Yearly`	Computes the start of year (1st of the current year) of the given date/time

     ```
     C#
     _consolidator = new DynamicDataConsolidator(Calendar.Weekly);
     // Alias:
     // _consolidator = CreateConsolidator(Calendar.Weekly, typeof(<dataType>));
     ```

   - Custom Periods

     If you need something more specific than the preceding time periods, define a method to set the start time and period of the consolidated bars. The method should receive a `DateTime` object that's based in the data time zone and should return a `CalendarInfo` object, which contains the start time of the bar in the data time zone and the duration of the consolidation period. The following example demonstrates how to create a custom consolidator for weekly bars:

```C#
_consolidator = new DynamicDataConsolidator(datetime => {
    var period = TimeSpan.FromDays(7);

    var timeSpan = new TimeSpan(17, 0, 0);
    var newDateTime = datetime.Date + timeSpan;
    var delta = 1 + (int)newDateTime.DayOfWeek;
    if (delta > 6)
    {
        delta = 0;
    }
    var start = newDateTime.AddDays(-delta);

    return new CalendarInfo(start, period);
});
```

2. Add an event handler to the consolidator.

```C#
_consolidator.DataConsolidated += ConsolidationHandler;
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

```C#
void ConsolidationHandler(object sender, <dataType> consolidatedBar)
{

}
```

If you use a custom consolidation period method, LEAN passes the consolidated bar to the consolidation handler when the consolidation period ends. The `Time` and `EndTime` properties of the consolidated bar reflect the data time zone, but the `Time` property of the algorithm still reflects the algorithm time zone .

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the data subscription, call the `AddConsolidator` method of the Subscription Manager.

```C#
SubscriptionManager.AddConsolidator(_symbol, _consolidator);
```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `<dataType>` object. You can update the consolidator

with data from the Slice object in the `OnData` method or with data from a history request .

```
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    if slice.ContainsKey(_symbol)
    {
        _consolidator.Update(slice[_symbol]);
    }
}

// Example 2: Update the consolidator with data from a history request
var history = History<<dataType>>(_symbol, 30, Resolution.Daily);
foreach (var data in history)
{
    _consolidator.Update(data);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

You can also use the `Consolidate` helper method to create calendar consolidators and register them for automatic updates.

```
_consolidator = Consolidate(_symbol, Calendar.Weekly, ConsolidationHandler);
```

If you use the `Consolidate` helper method, the consolidation handler doesn't receive an `object` argument.

```
void ConsolidationHandler(<dataType> consolidatedBar)
{

}
```

Reset Consolidators

To reset a consolidator, call its `Reset` method.

```
_consolidator.Reset();
```

If you are live trading Equities or backtesting Equities without the adjusted data normalization mode , reset your consolidators when splits and dividends occur. When a split or dividend occurs while the consolidator is in the process of building a bar, the open, high, and low may reflect prices from before the split or dividend. To avoid issues, call the consolidator's `Reset` method and then warm it up with `ScaledRaw` data from a history request .

```C#
public override void OnData(Slice data)
{
    // When a split or dividend occurs...
    if ((data.Splits.ContainsKey(_symbol) && data.Splits[_symbol].Type == SplitType.SplitOccurred) ||
        data.Dividends.ContainsKey(_symbol))
    {
        // If the consolidator is working on a bar...
        if (_consolidator.WorkingData != null)
        {
            // Get adjusted prices for the time period of the working bar.
            var history = History<TradeBar>(_symbol, _consolidator.WorkingData.Time, Time, dataNormalizationMode:
DataNormalizationMode.ScaledRaw);
            // Reset the consolidator.
            _consolidator.Reset();
            // Warm-up the consolidator with the adjusted price data.
            foreach (var bar in history)
            {
                _consolidator.Update(bar);
            }
        }
    }
}
```

Examples

The following examples demonstrate some common practices for calendar consolidators.

Example 1: Daily Futures Bars

Daily bars from the CME and Yahoo Finance represent the price action from 5 PM Central Time (CT) to 4 PM CT on the following day. In contrast, the regular daily resolution bars on QuantConnect represent the price action from 12 AM Eastern Time (ET) to 12 AM ET the following day. To create daily bars that represent the same period of time as the daily bars from the CME, the following algorithm consolidates minute bars (during regular and extended market hours) into daily bars that span from 5 PM CT to 4 PM CT on the following day. The algorithm then uses these consolidated daily bars to place take profit and stop loss orders as a function of the daily Average True Range .

```C#
public class CalendarConsolidatorExampleAlgorithm : QCAlgorithm
{
    private Future _future;
    private decimal _takeProfitPrice, _stopLossPrice;
    // Create an ATR indicator to set take profit and stop loss levels.
    private AverageTrueRange _atr = new(10);
    // Create a day tracker to avoid over-trading.
    private int _day = -1;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add ES Futures data and configure the continuous contract settings
        // so you can update the ATR with a smooth feed of prices.
        _future = AddFuture(
            Futures.Indices.SP500EMini,
            extendedMarketHours: true,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            dataMappingMode: DataMappingMode.OpenInterest,
            contractDepthOffset: 0
        );
        // Create a calendar consolidator to generate daily ES Futures bars
        // since market starts at 6pm EST to 5pm EST next day.
        var consolidator = new TradeBarConsolidator(FutureTradingHour);
        consolidator.DataConsolidated += OnConsolidated;
        // Warm up the consolidator and ATR indicator.
        var history = History<TradeBar>(_future.Symbol, 14500, Resolution.Minute);
        foreach (var bar in history)
        {
            consolidator.Update(bar);
        }
        // Subscribe the consolidator for automatic updates with ES data.
```

```
        SubscriptionManager.AddConsolidator(_future.Symbol, consolidator);
    }

    private CalendarInfo FutureTradingHour(DateTime datetime)
    {
        // Set the open time of the bar to be 6 PM EST.
        var start = datetime.Date.AddHours(datetime.Hour < 18 ? -6 : 18);
        // Set the close time of the bar to be 5 PM EST the next day, which is 23 hours later.
        return new CalendarInfo(start, TimeSpan.FromHours(23));
    }

    private void OnConsolidated(object sender, TradeBar bar)
    {
        // Update the ATR indicator since we want the daily ATR to set the stop price.
        _atr.Update(bar);
        // Set the daily take profit and stop price levels as a function of the previous
        // close price and the ATR.
        _takeProfitPrice = bar.Close + _atr * 2m;
        _stopLossPrice = bar.Close - _atr * 2m;
    }

    public override void OnData(Slice slice)
    {
        // Liquidate the current contract in the continuous contract series
        // when it's about to expire.
        if (slice.SymbolChangedEvents.TryGetValue(_future.Symbol, out var changedEvent))
        {
            Liquidate(changedEvent.OldSymbol);
        }

        // If we're not invested, buy the current contract in the continuous contract series.
        if (!Portfolio.Invested && Securities[_future.Mapped].Price != 0 && _day != slice.Time.Day)
        {
            MarketOrder(_future.Mapped, 1);
        }
        // If we're already invested, check if we've hit the TP/SL level.
        else if (Portfolio.Invested && slice.Bars.TryGetValue(_future.Symbol, out var bar))
        {
            // Check if the current price is above profit taker or below stop loss.
            if (bar.Close > _takeProfitPrice || bar.Close < _stopLossPrice)
            {
                // Exit the position.
                Liquidate();
            }
        }
        // Update the daily tracker so we only update our position once per day.
        _day = slice.Time.Day;
    }
}
```

Example 2: Create Daily `QuoteBar` Objects That Start at 5 PM

The following algorithm consolidates minute resolution Forex data into daily `QuoteBar` objects that open and close at 5 PM Eastern Time.

C#

```
public class CalendarConsolidatorExampleAlgorithm : QCAlgorithm
{
    private dynamic _pair;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add the USDJPY Forex pair.
        _pair = AddForex("USDJPY");
        // Create an EMA indicator for the pair.
        _pair.Ema = new ExponentialMovingAverage(10);
        // Create a QuoteBar consolidator with a custom consolidation period.
        var consolidator = new QuoteBarConsolidator(DailyForexConsolidationPeriod);
        // Attach a consolidation handler that will receive the consolidated bars.
        consolidator.DataConsolidated += ConsolidationHandler;
        // Subscribe the consolidator for automatic updates with the prices of the pair.
        SubscriptionManager.AddConsolidator(_pair.Symbol, consolidator);
        // Register the indicator for automatic updates with the consolidated bars.
        RegisterIndicator<IndicatorDataPoint>(_pair.Symbol, _pair.Ema, consolidator);
        // Warm up the consolidator and indicator.
        var history = History<QuoteBar>(_pair.Symbol, 29000, Resolution.Minute);
        foreach (var bar in history)
        {
            consolidator.Update(bar);
        }
    }

    // Define the consolidation period.
    private CalendarInfo DailyForexConsolidationPeriod(DateTime dt)
    {
        // Set the start of the bar to be 5 PM ET.
        var start = dt.Date;
        if (dt.Hour < 17)
        {
            start = start.AddHours(-7);
        }
        else
        {
            start = start.AddHours(17);
        }
        // Set the end of the bar to be 5 PM ET the next day.
        return new CalendarInfo(start, TimeSpan.FromDays(1));
    }

    private void ConsolidationHandler(object sender, QuoteBar consolidatedBar)
    {
        // Wait until the indicator is ready and the algorithm is running.
        if (!_pair.Ema.IsReady || IsWarmingUp)
        {
            return;
        }
        // Plot the closing price and the EMA.
        Plot(consolidatedBar.Symbol.ToString(), "Close", consolidatedBar.Close);
        Plot(consolidatedBar.Symbol.ToString(), "EMA", _pair.Ema.Current.Value);
    }
}
```

For a version of this algorithm with Time Period Consolidators, see Time Period Consolidators > Examples .

Example 3: Monthly Candlesticks

The following algorithm consolidates minute-resolution data into monthly `TradeBar` objects and then plots them as candlesticks .

C#

```
public class CalendarConsolidatorExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add an asset.
        var symbol = AddEquity("SPY").Symbol;
        // Create a consolidator that produces monthly TradeBar objects.
        // This is the stardard way:
        //var consolidator = new TradeBarConsolidator(Calendar.Monthly);
        // However, you can acomplish the same thing with a custom period:
        var consolidator = new TradeBarConsolidator(CustomMonthly);
        // Add a consolidation handler that plots the monthly bars as candlesticks.
        consolidator.DataConsolidated += (_, bar) => Plot(
            "Monthly Candlestick", symbol.Value, bar.Open, bar.High, bar.Low, bar.Close
        );
        // Register the consolidator for automatic updates.
        SubscriptionManager.AddConsolidator(symbol, consolidator);
    }

    public CalendarInfo CustomMonthly(DateTime dt)
    {
        // Set the start time to the beginning of the month.
        var start = dt.AddDays(1 - dt.Day).Date;
        // Set the end time to the beginning of next month.
        var end = start.AddMonths(1);
        return new CalendarInfo(start, (end - start));
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
DataConsolidationAlgorithm.cs C#

Consolidator Types

Count Consolidators

Introduction

Count consolidators aggregate data based on a number of bars or ticks. This type of consolidator aggregates n samples together, regardless of the time period the samples cover. Managing count consolidators is similiar to managing time period consolidators , except you create count consolidators with an integer argument instead of a time-based argument.

Consolidate Trade Bars

`TradeBar` consolidators aggregate `TradeBar` objects into `TradeBar` objects of the same size or larger. Follow these steps to create and manage a `TradeBar` consolidator based on a number of samples:

1. Create the consolidator.

   To create a count consolidator, pass the number of samples to the consolidator constructor.

   ```
   C#
   _consolidator = new TradeBarConsolidator(10);
   ```

2. Add an event handler to the consolidator.

   ```
   C#
   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

   ```
   C#
   void ConsolidationHandler(object sender, TradeBar consolidatedBar)
   {

   }
   ```

   When the consolidator receives the n -th data point, it passes the consolidated bar to the event handler.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

   - Automatic Updates

     To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
SubscriptionManager.AddConsolidator(_symbol, _consolidator);
```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `TradeBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

  ```
  // Example 1: Update the consolidator with data from the Slice object
  public override void OnData(Slice slice)
  {
      var tradeBar = slice.Bars[_symbol];
      _consolidator.Update(tradeBar);
  }

  // Example 2: Update the consolidator with data from a history request
  var history = History<TradeBar>(_symbol, 30, Resolution.Minute);
  foreach (var tradeBar in history)
  {
      _consolidator.Update(tradeBar);
  }
  ```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Quote Bars

`QuoteBar` consolidators aggregate `QuoteBar` objects into `QuoteBar` objects of the same size or larger. Follow these steps to create and manage a `QuoteBar` consolidator based on a number of samples:

1. Create the consolidator.

   To create a count consolidator, pass the number of samples to the consolidator constructor.

   ```
   _consolidator = new QuoteBarConsolidator(10);
   ```

2. Add an event handler to the consolidator.

   ```
   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself.

Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

```
C#
void ConsolidationHandler(object sender, QuoteBar consolidatedBar)
{

}
```

When the consolidator receives the n -th data point, it passes the consolidated bar to the event handler.

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
C#
SubscriptionManager.AddConsolidator(_symbol, _consolidator);
```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `QuoteBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
C#
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var quoteBar = slice.QuoteBars[_symbol];
    _consolidator.Update(quoteBar);
}

// Example 2: Update the consolidator with data from a history request
var history = History<QuoteBar>(_symbol, 30, Resolution.Minute);
foreach (var quoteBar in history)
{
    _consolidator.Update(quoteBar);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
C#
SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Trade Ticks

`Tick` consolidators aggregate `Tick` objects into `TradeBar` objects. Follow these steps to create and manage a `Tick` consolidator based on a number of samples:

1. Create the consolidator.

   To create a count consolidator, pass the number of samples to the consolidator constructor.

   ```
   C#
   _consolidator = new TickConsolidator(10);
   ```

2. Add an event handler to the consolidator.

   ```
   C#
   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

   ```
   C#
   void ConsolidationHandler(object sender, TradeBar consolidatedBar)
   {

   }
   ```

   When the consolidator receives the n -th data point, it passes the consolidated bar to the event handler.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

   - Automatic Updates

     To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

     ```
     C#
     SubscriptionManager.AddConsolidator(_symbol, _consolidator);
     ```

   - Manual Updates

     Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var ticks = slice.Ticks[_symbol];
    foreach (var tick in ticks)
    {
     _consolidator.Update(tick);
    }
}

// Example 2: Update the consolidator with data from a history request
var ticks = History<Tick>(_symbol, TimeSpan.FromMinutes(3), Resolution.Tick);
foreach (var tick in ticks)
{
    _consolidator.Update(tick);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Quote Ticks

`Tick` quote bar consolidators aggregate `Tick` objects that represent quotes into `QuoteBar` objects. Follow these steps to create and manage a `Tick` quote bar consolidator based on a number of samples:

1. Create the consolidator.

   To create a count consolidator, pass the number of samples to the consolidator constructor.

   ```
   _consolidator = new TickQuoteBarConsolidator(10);
   ```

2. Add an event handler to the consolidator.

   ```
   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler`, not `ConsolidationHandler()`.

3. Define the consolidation handler.

```
void ConsolidationHandler(object sender, QuoteBar consolidatedBar)
{

}
```

When the consolidator receives the n -th data point, it passes the consolidated bar to the event handler.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

   - Automatic Updates

     To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

     ```
     SubscriptionManager.AddConsolidator(_symbol, _consolidator);
     ```

   - Manual Updates

     Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

     ```
     // Example 1: Update the consolidator with data from the Slice object
     public override void OnData(Slice slice)
     {
         var ticks = slice.Ticks[_symbol];
         foreach (var tick in ticks)
         {
          _consolidator.Update(tick);
         }
     }

     // Example 2: Update the consolidator with data from a history request
     var ticks = History<Tick>(_symbol, TimeSpan.FromMinutes(3), Resolution.Tick);
     foreach (var tick in ticks)
     {
         _consolidator.Update(tick);
     }
     ```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Other Data

`<dataType>` consolidators aggregate various types of data objects into `<dataType>` objects of the same size or larger. If you

consolidate custom data or alternative datasets, check the definition of the data class to see its data type. Follow these steps to create and manage a `<dataType>` consolidator based on a number of samples:

1. Create the consolidator.

   To create a count consolidator, pass the number of samples to the consolidator constructor.

   ```
   C#
   _consolidator = new DynamicDataConsolidator(10);
   ```

2. Add an event handler to the consolidator.

   ```
   C#
   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

   ```
   C#
   void ConsolidationHandler(object sender, <dataType> consolidatedBar)
   {

   }
   ```

   When the consolidator receives the n -th data point, it passes the consolidated bar to the event handler.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

   - Automatic Updates

     To automatically update a consolidator with data from the data subscription, call the `AddConsolidator` method of the Subscription Manager.

     ```
     C#
     SubscriptionManager.AddConsolidator(_symbol, _consolidator);
     ```

   - Manual Updates

     Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `<dataType>` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    if slice.ContainsKey(_symbol)
    {
        _consolidator.Update(slice[_symbol]);
    }
}

// Example 2: Update the consolidator with data from a history request
var history = History<<dataType>>(_symbol, 30, Resolution.Daily);
foreach (var data in history)
{
    _consolidator.Update(data);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Reset Consolidators

To reset a consolidator, call its `Reset` method.

```
_consolidator.Reset();
```

If you are live trading Equities or backtesting Equities without the adjusted data normalization mode , reset your consolidators when splits and dividends occur. When a split or dividend occurs while the consolidator is in the process of building a bar, the open, high, and low may reflect prices from before the split or dividend. To avoid issues, call the consolidator's `Reset` method and then warm it up with `ScaledRaw` data from a history request .

```
public override void OnData(Slice data)
{
    // When a split or dividend occurs...
    if ((data.Splits.ContainsKey(_symbol) && data.Splits[_symbol].Type == SplitType.SplitOccurred) ||
        data.Dividends.ContainsKey(_symbol))
    {
        // If the consolidator is working on a bar...
        if (_consolidator.WorkingData != null)
        {
            // Get adjusted prices for the time period of the working bar.
            var history = History<TradeBar>(_symbol, _consolidator.WorkingData.Time, Time, dataNormalizationMode:
DataNormalizationMode.ScaledRaw);
            // Reset the consolidator.
            _consolidator.Reset();
            // Warm-up the consolidator with the adjusted price data.
            foreach (var bar in history)
            {
                _consolidator.Update(bar);
            }
        }
    }
}
```

Examples

The following examples demonstrate common practices for implementing count consolidator.

Example 1: Tick Count Consolidator

```C#
public class CountConsolidatorAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    // Create EMA indicators for trade signal generation.
    private ExponentialMovingAverage _ema = new(14);
    // The _day variable controls trade daily when open.
    private int _day = -1;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 4);
        SetEndDate(2024, 9, 5);

        // Request SPY data to feed to indicators and trade.
        _spy = AddEquity("SPY", Resolution.Tick).Symbol;

        // Create a 10000 trade tick consolidator to feed the "volume" bar to the EMA indicator for trend entry/exit
signal.
        var tickConsolidator = new TickConsolidator(10000);

        // Subscribe the consolidators to SPY data to automatically update the indicators.
        RegisterIndicator(_spy, _ema, tickConsolidator);

        SetWarmUp(120);
    }

    public override void OnData(Slice slice)
    {
        if (_ema.IsReady && slice.Ticks.TryGetValue(_spy, out var ticks) && slice.Time.Day != _day)
        {
            // Obtain the latest quote price from the latest tick.
            var lastTick = ticks.MaxBy(x => x.EndTime);

            // Trade the trend by EMA crosses.
            if (lastTick.Price > _ema && !Portfolio[_spy].IsLong)
            {
                SetHoldings(_spy, 0.5m);
            }
            else if (lastTick.Price < _ema && !Portfolio[_spy].IsShort)
            {
                SetHoldings(_spy, -0.5m);
            }

            _day = slice.Time.Day;
        }
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
DataConsolidationAlgorithm.cs C#

Consolidator Types

Mixed-Mode Consolidators

Introduction

Mixed-mode consolidators are a combination of count consolidators and time period consolidators. This type of consolidator aggregates n samples together or aggregates samples over a specific time period, whichever happens first.

Consolidate Trade Bars

`TradeBar` consolidators aggregate `TradeBar` objects into `TradeBar` objects of the same size or larger. Follow these steps to create and manage a `TradeBar` consolidator based on a period of time or a number of samples, whichever occurs first:

1. Create the consolidator.

   To create a mixed-mode consolidator, pass a `TimeSpan` object and an integer to the consolidator constructor. The consolidator time period must be greater than or equal to the resolution of the security subscription. For instance, you can aggregate minute bars into 10-minute bars, but you can't aggregate hour bars into 10-minute bars.

   ```
   C#
   _consolidator = new TradeBarConsolidator(10, TimeSpan.FromDays(1));
   ```

   The time period is relative to the data time zone , not the algorithm time zone . If you consolidate Crypto data into daily bars, the event handler receives the consolidated bars at midnight 12:00 AM Coordinated Universal Time (UTC), regardless of the algorithm time zone.

   If you consolidate on an hourly basis, the consolidator ends at the top of the hour, not every hour after the market open. For US Equities, that's 10 AM Eastern Time (ET), not 10:30 AM ET. To change this behavior, set the `startTime` parameter of the =$consolidatorClassName?> type constructor to `TimeSpan(=$this->timeSpanStartTime?>)`

2. Add an event handler to the consolidator.

   ```
   C#
   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

   ```
   C#
   void ConsolidationHandler(object sender, TradeBar consolidatedBar)
   {

   }
   ```

The consolidation event handler receives bars when the consolidated bar closes based on the data time zone or the number of samples, whichever occurs first. If you subscribe to minute resolution data for Bitcoin and create an hourly consolidator, you receive consolidated bars at the top of each hour. However, if you subscribe to minute resolution data for the regular trading hours of US Equities and create a daily consolidator, you receive consolidated bars at market close or midnight, depending on your `DailyPreciseEndTime` setting .

4. Update the consolidator.

   You can automatically or manually update the consolidator.

   - Automatic Updates

     To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

     ```
     C#

     SubscriptionManager.AddConsolidator(_symbol, _consolidator);
     ```

   - Manual Updates

     Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `TradeBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

     ```
     C#

     // Example 1: Update the consolidator with data from the Slice object
     public override void OnData(Slice slice)
     {
         var tradeBar = slice.Bars[_symbol];
         _consolidator.Update(tradeBar);
     }

     // Example 2: Update the consolidator with data from a history request
     var history = History<TradeBar>(_symbol, 30, Resolution.Minute);
     foreach (var tradeBar in history)
     {
         _consolidator.Update(tradeBar);
     }
     ```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   C#

   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Quote Bars

`QuoteBar` consolidators aggregate `QuoteBar` objects into `QuoteBar` objects of the same size or larger. Follow these steps to create and manage a `QuoteBar` consolidator based on a period of time or a number of samples, whichever occurs first:

1. Create the consolidator.

To create a mixed-mode consolidator, pass a `TimeSpan` object and an integer to the consolidator constructor. The consolidator time period must be greater than or equal to the resolution of the security subscription. For instance, you can aggregate minute bars into 10-minute bars, but you can't aggregate hour bars into 10-minute bars.

```
C#
_consolidator = new QuoteBarConsolidator(10, TimeSpan.FromDays(1));
```

The time period is relative to the data time zone , not the algorithm time zone . If you consolidate Crypto data into daily bars, the event handler receives the consolidated bars at midnight 12:00 AM Coordinated Universal Time (UTC), regardless of the algorithm time zone.

If you consolidate on an hourly basis, the consolidator ends at the top of the hour, not every hour after the market open. For US Equities, that's 10 AM Eastern Time (ET), not 10:30 AM ET. To change this behavior, set the `startTime` parameter of the =$consolidatorClassName?> type constructor to `TimeSpan(=$this->timeSpanStartTime?>)`

2. Add an event handler to the consolidator.

```
C#
_consolidator.DataConsolidated += ConsolidationHandler;
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

```
C#
void ConsolidationHandler(object sender, QuoteBar consolidatedBar)
{

}
```

The consolidation event handler receives bars when the consolidated bar closes based on the data time zone or the number of samples, whichever occurs first. If you subscribe to minute resolution data for Bitcoin and create an hourly consolidator, you receive consolidated bars at the top of each hour. However, if you subscribe to minute resolution data for the regular trading hours of US Equities and create a daily consolidator, you receive consolidated bars at market close or midnight, depending on your `DailyPreciseEndTime` setting .

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  C#
  SubscriptionManager.AddConsolidator(_symbol, _consolidator);
  ```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `QuoteBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

  ```
  C#

  // Example 1: Update the consolidator with data from the Slice object
  public override void OnData(Slice slice)
  {
      var quoteBar = slice.QuoteBars[_symbol];
      _consolidator.Update(quoteBar);
  }

  // Example 2: Update the consolidator with data from a history request
  var history = History<QuoteBar>(_symbol, 30, Resolution.Minute);
  foreach (var quoteBar in history)
  {
      _consolidator.Update(quoteBar);
  }
  ```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   C#

   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Trade Ticks

`Tick` consolidators aggregate `Tick` objects into `TradeBar` objects. Follow these steps to create and manage a `Tick` consolidator based on a period of time or a number of samples, whichever occurs first:

1. Create the consolidator.

   To create a mixed-mode consolidator, pass a `TimeSpan` object and an integer to the consolidator constructor. The consolidator time period must be greater than or equal to the resolution of the security subscription. For instance, you can aggregate minute bars into 10-minute bars, but you can't aggregate hour bars into 10-minute bars.

   ```
   C#

   _consolidator = new TickConsolidator(10, TimeSpan.FromMilliseconds(100));
   ```

   The time period is relative to the data time zone , not the algorithm time zone . If you consolidate Crypto data into daily bars, the event handler receives the consolidated bars at midnight 12:00 AM Coordinated Universal Time (UTC), regardless of the algorithm time zone.

   If you consolidate on an hourly basis, the consolidator ends at the top of the hour, not every hour after the market open. For US Equities, that's 10 AM Eastern Time (ET), not 10:30 AM ET. To change this behavior, set the `startTime` parameter of the =$consolidatorClassName?> type constructor to `TimeSpan(=$this->timeSpanStartTime?>)`

2. Add an event handler to the consolidator.

```
_consolidator.DataConsolidated += ConsolidationHandler;
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler`, not `ConsolidationHandler()`.

3. Define the consolidation handler.

```
void ConsolidationHandler(object sender, TradeBar consolidatedBar)
{

}
```

The consolidation event handler receives bars when the consolidated bar closes based on the data time zone or the number of samples, whichever occurs first. If you subscribe to tick resolution data for Bitcoin and create an hourly consolidator, you receive consolidated bars at the top of each hour. However, if you subscribe to tick resolution data for the regular trading hours of US Equities and create a daily consolidator, you receive consolidated bars at market close or midnight, depending on your `DailyPreciseEndTime` setting.

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
SubscriptionManager.AddConsolidator(_symbol, _consolidator);
```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period, you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request.

```
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var ticks = slice.Ticks[_symbol];
    foreach (var tick in ticks)
    {
     _consolidator.Update(tick);
    }
}

// Example 2: Update the consolidator with data from a history request
var ticks = History<Tick>(_symbol, TimeSpan.FromMinutes(3), Resolution.Tick);
foreach (var tick in ticks)
{
    _consolidator.Update(tick);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Quote Ticks

`Tick` quote bar consolidators aggregate `Tick` objects that represent quotes into `QuoteBar` objects. Follow these steps to create and manage a `Tick` quote bar consolidator based on a period of time or a number of samples, whichever occurs first:

1. Create the consolidator.

   To create a mixed-mode consolidator, pass a `TimeSpan` object and an integer to the consolidator constructor. The consolidator time period must be greater than or equal to the resolution of the security subscription. For instance, you can aggregate minute bars into 10-minute bars, but you can't aggregate hour bars into 10-minute bars.

   ```
   _consolidator = new TickQuoteBarConsolidator(10, TimeSpan.FromMilliseconds(100));
   ```

   The time period is relative to the data time zone , not the algorithm time zone . If you consolidate Crypto data into daily bars, the event handler receives the consolidated bars at midnight 12:00 AM Coordinated Universal Time (UTC), regardless of the algorithm time zone.

   If you consolidate on an hourly basis, the consolidator ends at the top of the hour, not every hour after the market open. For US Equities, that's 10 AM Eastern Time (ET), not 10:30 AM ET. To change this behavior, set the `startTime` parameter of the =$consolidatorClassName?> type constructor to `TimeSpan(=$this->timeSpanStartTime?>)`

2. Add an event handler to the consolidator.

   ```
   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating

consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

```
C#

void ConsolidationHandler(object sender, QuoteBar consolidatedBar)
{

}
```

The consolidation event handler receives bars when the consolidated bar closes based on the data time zone or the number of samples, whichever occurs first. If you subscribe to tick resolution data for Bitcoin and create an hourly consolidator, you receive consolidated bars at the top of each hour. However, if you subscribe to tick resolution data for the regular trading hours of US Equities and create a daily consolidator, you receive consolidated bars at market close or midnight, depending on your `DailyPreciseEndTime` setting .

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
C#

SubscriptionManager.AddConsolidator(_symbol, _consolidator);
```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
C#

// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var ticks = slice.Ticks[_symbol];
    foreach (var tick in ticks)
    {
     _consolidator.Update(tick);
    }
}

// Example 2: Update the consolidator with data from a history request
var ticks = History<Tick>(_symbol, TimeSpan.FromMinutes(3), Resolution.Tick);
foreach (var tick in ticks)
{
    _consolidator.Update(tick);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Other Data

<dataType> consolidators aggregate various types of data objects into <dataType> objects of the same size or larger. If you consolidate custom data or alternative datasets, check the definition of the data class to see its data type. Follow these steps to create and manage a <dataType> consolidator based on a period of time or a number of samples, whichever occurs first:

1. Create the consolidator.

   To create a mixed-mode consolidator, pass a TimeSpan object and an integer to the consolidator constructor. The consolidator time period must be greater than or equal to the resolution of the security subscription. For instance, you can aggregate minute bars into 10-minute bars, but you can't aggregate hour bars into 10-minute bars.

   ```
   _consolidator = new DynamicDataConsolidator(10, TimeSpan.FromDays(1));
   ```

   The time period is relative to the data time zone , not the algorithm time zone . If you consolidate Crypto data into daily bars, the event handler receives the consolidated bars at midnight 12:00 AM Coordinated Universal Time (UTC), regardless of the algorithm time zone.

   If you consolidate on an hourly basis, the consolidator ends at the top of the hour, not every hour after the market open. For US Equities, that's 10 AM Eastern Time (ET), not 10:30 AM ET. To change this behavior, set the startTime parameter of the =$consolidatorClassName?> type constructor to TimeSpan(=$this->timeSpanStartTime?>)

2. Add an event handler to the consolidator.

   ```
   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis () at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be ConsolidationHandler , not ConsolidationHandler() .

3. Define the consolidation handler.

   ```
   void ConsolidationHandler(object sender, <dataType> consolidatedBar)
   {

   }
   ```

   The consolidation event handler receives bars when the consolidated bar closes based on the data time zone or the number of samples, whichever occurs first.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

   - Automatic Updates

     To automatically update a consolidator with data from the data subscription, call the `AddConsolidator` method of the Subscription Manager.

     ```
     C#

     SubscriptionManager.AddConsolidator(_symbol, _consolidator);
     ```

   - Manual Updates

     Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `<dataType>` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

     ```
     C#

     // Example 1: Update the consolidator with data from the Slice object
     public override void OnData(Slice slice)
     {
         if slice.ContainsKey(_symbol)
         {
             _consolidator.Update(slice[_symbol]);
         }
     }

     // Example 2: Update the consolidator with data from a history request
     var history = History<<dataType>>(_symbol, 30, Resolution.Daily);
     foreach (var data in history)
     {
         _consolidator.Update(data);
     }
     ```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   C#

   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Reset Consolidators

To reset a consolidator, call its `Reset` method.

```
C#

_consolidator.Reset();
```

If you are live trading Equities or backtesting Equities without the adjusted data normalization mode , reset your consolidators when splits and dividends occur. When a split or dividend occurs while the consolidator is in the process of building a bar, the open, high, and low may reflect prices from before the split or dividend. To avoid issues, call the consolidator's `Reset` method

and then warm it up with `ScaledRaw` data from a history request .

```C#
public override void OnData(Slice data)
{
    // When a split or dividend occurs...
    if ((data.Splits.ContainsKey(_symbol) && data.Splits[_symbol].Type == SplitType.SplitOccurred) ||
        data.Dividends.ContainsKey(_symbol))
    {
        // If the consolidator is working on a bar...
        if (_consolidator.WorkingData != null)
        {
            // Get adjusted prices for the time period of the working bar.
            var history = History<TradeBar>(_symbol, _consolidator.WorkingData.Time, Time, dataNormalizationMode:
DataNormalizationMode.ScaledRaw);
            // Reset the consolidator.
            _consolidator.Reset();
            // Warm-up the consolidator with the adjusted price data.
            foreach (var bar in history)
            {
                _consolidator.Update(bar);
            }
        }
    }
}
```

Examples

The following examples demonstrate common practices for implementing mixed-mode consolidators.

Example 1: Opening Range Breakout

The following algorithm demonstrates an opening range breakout strategy for BJK (a Gaming ETF). We create a 15-count/30-minute trade bar as the opening range for signal generation. If the close price at 10 am is higher than the high price of the aggregated opening bar, we long BJK and exit by a 2% trailing stop loss.

C#

```
public class MixedModeConsolidatorAlgorithm : QCAlgorithm
{
    private Symbol _equity;
    private TradeBarConsolidator _consolidator;
    private decimal _highPrice = 0m;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request CARZ data to feed to the consolidators and trade.
        _equity = AddEquity("CARZ", Resolution.Minute).Symbol;

        // Create a 10-count-or-20-minute trade bar consolidator to create the opening range for trade signal
generation.
        // CARZ is a less liquid ETF, so we use a mixed-mode consolidator.
        _consolidator = new TradeBarConsolidator(10, TimeSpan.FromMinutes(20));

        // Subscribe the consolidators to CARZ data for automatic updating.
        SubscriptionManager.AddConsolidator(_equity, _consolidator);

        // We check if the entry condition is attained 30 minutes after the market opens.
        Schedule.On(
            DateRules.EveryDay(_equity),
            TimeRules.AfterMarketOpen(_equity, 30),
            Entry
        );

        SetWarmUp(1);
    }

    private void Entry()
    {
        // Check if the entry condition is met: the current price is higher than the high price of the open range.
        var current = Securities[_equity].Close;
        if (current > (_consolidator.Consolidated as TradeBar).High)
        {
            MarketOrder(_equity, 20);
            // Cache the current price for trailing stop loss need.
            _highPrice = current;
        }
    }

    public override void OnData(Slice slice)
    {
        if (slice.Bars.TryGetValue(_equity, out var bar) && Portfolio.Invested)
        {
            // Update the high price if the current high is higher to update the trailing stop price.
            if (bar.High > _highPrice)
            {
                _highPrice = bar.High;
            }

            // Exit position if the trailing stop loss is hit.
            if (bar.Close < _highPrice * 0.98m)
            {
                Liquidate();
                _highPrice = 0m;
            }
        }
    }
}
```

Consolidator Types

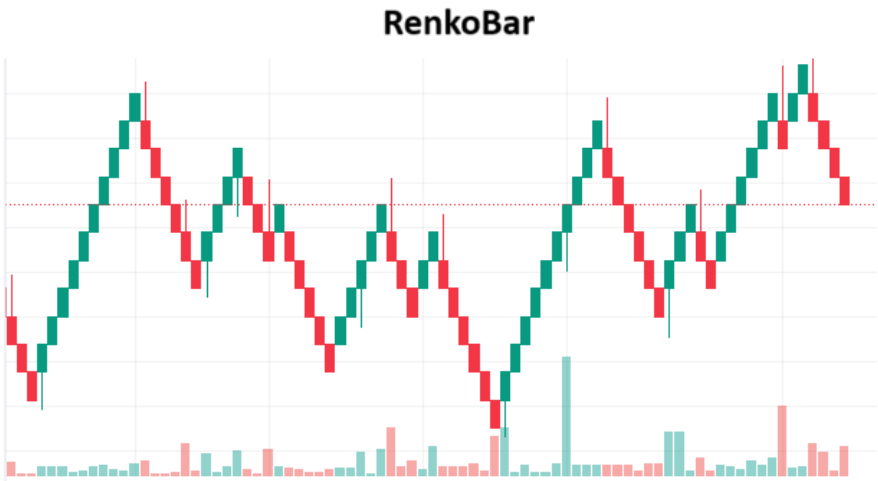
Renko Consolidators

Renko Consolidators

Renko Consolidators

Introduction

Most Renko consolidators aggregate bars based on a fixed price movement. The `RenkoConsolidator` produces Renko bars by their traditional definition. In the case of a $1 bar size, the `RenkoConsolidator` produces bars that have a body spanning 1.*Theopeningpriceofthefirstbarissettotheclosest*1 multiple of the first trade. When the price moves by at least 1, *thefirstbarcloses*. *Ifabarisarisingbar*, *thefollowingbarcloseswhenthepricemoves*1 above the closing price of the previous bar or 1*belowtheopeningpriceofthepreviousbar*. *Ifabarisafallingbar*, *thefollowingbarcloseswhenthepricemoves*1 below the closing price of the previous bar or $1 above the opening price of the previous bar. If the price jumps multiple dollars in a single tick, the `RenkoConsolidator` produces multiple $1 bars in a single time step.



Consolidate Trade Bars

`TradeBar` consolidators aggregate `TradeBar` objects into `RenkoBar` objects. Follow these steps to create and manage a `TradeBar` consolidator based on the traditional Renko bar rules:

1. Create the consolidator.

   To create a Renko consolidator, pass the bar size to the `RenkoConsolidator` constructor.

   ```
   // Create a Renko consolidator that emits a bar when the price moves $1
   _consolidator = new RenkoConsolidator(1m);
   ```

   C#

2. Add an event handler to the consolidator.

```
_consolidator.DataConsolidated += ConsolidationHandler;
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler`, not `ConsolidationHandler()`.

3. Define the consolidation handler.

```
void ConsolidationHandler(object sender, RenkoBar consolidatedBar)
{

}
```

The consolidation event handler receives bars when the price movement forms a new Renko bar.

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
SubscriptionManager.AddConsolidator(_symbol, _consolidator);
```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period, you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `TradeBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request.

```
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var tradeBar = slice.Bars[_symbol];
    _consolidator.Update(tradeBar);
}

// Example 2: Update the consolidator with data from a history request
var history = History<TradeBar>(_symbol, 30, Resolution.Minute);
foreach (var tradeBar in history)
{
    _consolidator.Update(tradeBar);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Quote Bars

`QuoteBar` consolidators aggregate `QuoteBar` objects into `RenkoBar` objects. Follow these steps to create and manage a `QuoteBar` consolidator based on the traditional Renko bar rules:

1. Create the consolidator.

   To create a Renko consolidator, pass the bar size to the `RenkoConsolidator` constructor.

   ```
   // Create a Renko consolidator that emits a bar when the price moves $1
   _consolidator = new RenkoConsolidator(1m);
   ```

2. Add an event handler to the consolidator.

   ```
   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler`, not `ConsolidationHandler()`.

3. Define the consolidation handler.

   ```
   void ConsolidationHandler(object sender, RenkoBar consolidatedBar)
   {

   }
   ```

   The consolidation event handler receives bars when the price movement forms a new Renko bar.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

   - Automatic Updates

     To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

     ```
     SubscriptionManager.AddConsolidator(_symbol, _consolidator);
     ```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `QuoteBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

  ```
  C#

  // Example 1: Update the consolidator with data from the Slice object
  public override void OnData(Slice slice)
  {
      var quoteBar = slice.QuoteBars[_symbol];
      _consolidator.Update(quoteBar);
  }

  // Example 2: Update the consolidator with data from a history request
  var history = History<QuoteBar>(_symbol, 30, Resolution.Minute);
  foreach (var quoteBar in history)
  {
      _consolidator.Update(quoteBar);
  }
  ```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   C#

   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Trade Ticks

`Tick` consolidators aggregate `Tick` objects into `RenkoBar` objects. Follow these steps to create and manage a `Tick` consolidator based on the traditional Renko bar rules:

1. Create the consolidator.

   To create a Renko consolidator, pass the bar size to the `RenkoConsolidator` constructor.

   ```
   C#

   // Create a Renko consolidator that emits a bar when the price moves $1
   _consolidator = new RenkoConsolidator(1m);
   ```

2. Add an event handler to the consolidator.

   ```
   C#

   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

```
C#
void ConsolidationHandler(object sender, RenkoBar consolidatedBar)
{

}
```

The consolidation event handler receives bars when the price movement forms a new Renko bar.

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
C#
SubscriptionManager.AddConsolidator(_symbol, _consolidator);
```

- Manual Updates

Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
C#
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var ticks = slice.Ticks[_symbol];
    foreach (var tick in ticks)
    {
     _consolidator.Update(tick);
    }
}

// Example 2: Update the consolidator with data from a history request
var ticks = History<Tick>(_symbol, TimeSpan.FromMinutes(3), Resolution.Tick);
foreach (var tick in ticks)
{
    _consolidator.Update(tick);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
C#
SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Quote Ticks

`Tick` quote bar consolidators aggregate `Tick` objects that represent quotes into `RenkoBar` objects. Follow these steps to create and manage a `Tick` quote bar consolidator based on the traditional Renko bar rules:

1. Create the consolidator.

   To create a Renko consolidator, pass the bar size to the `RenkoConsolidator` constructor.

   ```
   C#
   // Create a Renko consolidator that emits a bar when the price moves $1
   _consolidator = new RenkoConsolidator(1m);
   ```

2. Add an event handler to the consolidator.

   ```
   C#
   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

   ```
   C#
   void ConsolidationHandler(object sender, RenkoBar consolidatedBar)
   {

   }
   ```

   The consolidation event handler receives bars when the price movement forms a new Renko bar.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

   - Automatic Updates

     To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

     ```
     C#
     SubscriptionManager.AddConsolidator(_symbol, _consolidator);
     ```

   - Manual Updates

     Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```C#
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var ticks = slice.Ticks[_symbol];
    foreach (var tick in ticks)
    {
     _consolidator.Update(tick);
    }
}

// Example 2: Update the consolidator with data from a history request
var ticks = History<Tick>(_symbol, TimeSpan.FromMinutes(3), Resolution.Tick);
foreach (var tick in ticks)
{
    _consolidator.Update(tick);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```C#
   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Reset Consolidators

To reset a consolidator, call its `Reset` method.

```C#
_consolidator.Reset();
```

If you are live trading Equities or backtesting Equities without the adjusted data normalization mode , reset your consolidators when splits and dividends occur. When a split or dividend occurs while the consolidator is in the process of building a bar, the open, high, and low may reflect prices from before the split or dividend. To avoid issues, call the consolidator's `Reset` method and then warm it up with `ScaledRaw` data from a history request .

```C#
public override void OnData(Slice data)
{
    // When a split or dividend occurs...
    if ((data.Splits.ContainsKey(_symbol) && data.Splits[_symbol].Type == SplitType.SplitOccurred) ||
        data.Dividends.ContainsKey(_symbol))
    {
        // If the consolidator is working on a bar...
        if (_consolidator.WorkingData != null)
        {
            // Get adjusted prices for the time period of the working bar.
            var history = History<TradeBar>(_symbol, _consolidator.WorkingData.Time, Time, dataNormalizationMode:
DataNormalizationMode.ScaledRaw);
            // Reset the consolidator.
            _consolidator.Reset();
            // Warm-up the consolidator with the adjusted price data.
            foreach (var bar in history)
            {
                _consolidator.Update(bar);
            }
        }
    }
}
```

Renko Consolidators

Classic Renko Consolidators

Introduction

Most Renko consolidators aggregate bars based on a fixed price movement. The `ClassicRenkoConsolidator` produces a different type of Renko bars than the `RenkoConsolidator` . A `ClassicRenkoConsolidator` with a bar size of 1*producesanewbarthatspans*1 every time an asset closes $1 away from the close of the previous bar. If the price jumps multiple dollars in a single tick, the `ClassicRenkoConsolidator` only produces one bar per time step where the open of each bar matches the close of the previous bar.

Consolidate Trade Bars

`TradeBar` consolidators aggregate `TradeBar` objects into `RenkoBar` objects. Follow these steps to create and manage a `TradeBar` consolidator based on the preceding Renko bar rules:

1. Create the consolidator.

   To create a classic Renko consolidator, pass the bar size to the `ClassicRenkoConsolidator` constructor.

   ```
   C#
   // Create a Classic Renko consolidator that emits a bar when the price moves $1
   _consolidator = new ClassicRenkoConsolidator(1m);
   ```

   The `ClassicRenkoConsolidator` has the following default behavior:

   - It uses the `Value` property of the `IBaseData` object it receives to build the Renko bars
   - It ignores the volume of the input data
   - It enforces the open and close of each bar to be a multiple of the bar size

   To build the Renko bars with a different property than the `Value` of the `IBaseData` object, provide a `selector` argument. The `selector` should be a function that receives the `IBaseData` object and returns a decimal value.

   ```
   C#
   _consolidator = new ClassicRenkoConsolidator(1m, data => (data as TradeBar).High);
   ```

   To add a non-zero `Volume` property to the Renko bars, provide a `volumeSelector` argument. The `volumeSelector` should be a function that receives the `IBaseData` object and returns a decimal value.

   ```
   C#
   _consolidator = new ClassicRenkoConsolidator(1m, null, data => (data as TradeBar).Volume);
   ```

   To relax the requirement that the open and close of the Renko bars must be a multiple of bar size, disable the `evenBars` argument. If you disable `evenBars` , the open value of the first Renko bar is set to the first value from the `selector` . The following opening and closing Renko bar values are all multiples of the first value from the `selector`

```
_consolidator = new ClassicRenkoConsolidator(1m, evenBars: false);
```

2. Add an event handler to the consolidator.

```
_consolidator.DataConsolidated += ConsolidationHandler;
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

```
void ConsolidationHandler(object sender, RenkoBar consolidatedBar)
{

}
```

The consolidation event handler receives bars when the price movement forms a new classic Renko bar.

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  SubscriptionManager.AddConsolidator(_symbol, _consolidator);
  ```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `TradeBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var tradeBar = slice.Bars[_symbol];
    _consolidator.Update(tradeBar);
}

// Example 2: Update the consolidator with data from a history request
var history = History<TradeBar>(_symbol, 30, Resolution.Minute);
foreach (var tradeBar in history)
{
    _consolidator.Update(tradeBar);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Quote Bars

`QuoteBar` consolidators aggregate `QuoteBar` objects into `RenkoBar` objects. Follow these steps to create and manage a `QuoteBar` consolidator based on the preceding Renko bar rules:

1. Create the consolidator.

   To create a classic Renko consolidator, pass the bar size to the `ClassicRenkoConsolidator` constructor.

   ```
   // Create a Classic Renko consolidator that emits a bar when the price moves $1
   _consolidator = new ClassicRenkoConsolidator(1m);
   ```

   The `ClassicRenkoConsolidator` has the following default behavior:

   - It uses the `Value` property of the `IBaseData` object it receives to build the Renko bars
   - It ignores the volume of the input data
   - It enforces the open and close of each bar to be a multiple of the bar size

   The following arguments enable you to create Renko bars that aggregate the excess liquidity on the bid.

   ```
   _consolidator = new ClassicRenkoConsolidator(10, null,
       data => (data as QuoteBar).LastBidSize - (data as QuoteBar).LastAskSize);
   ```

   To relax the requirement that the open and close of the Renko bars must be a multiple of bar size, disable the `evenBars` argument. If you disable `evenBars` , the open value of the first Renko bar is set to the first value from the `selector` . The following opening and closing Renko bar values are all multiples of the first value from the `selector`

```
_consolidator = new ClassicRenkoConsolidator(1m, evenBars: false);
```

2. Add an event handler to the consolidator.

```
_consolidator.DataConsolidated += ConsolidationHandler;
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

```
void ConsolidationHandler(object sender, RenkoBar consolidatedBar)
{

}
```

The consolidation event handler receives bars when the price movement forms a new classic Renko bar.

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  SubscriptionManager.AddConsolidator(_symbol, _consolidator);
  ```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `QuoteBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```C#
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var quoteBar = slice.QuoteBars[_symbol];
    _consolidator.Update(quoteBar);
}

// Example 2: Update the consolidator with data from a history request
var history = History<QuoteBar>(_symbol, 30, Resolution.Minute);
foreach (var quoteBar in history)
{
    _consolidator.Update(quoteBar);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```C#
   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Trade Ticks

`Tick` consolidators aggregate `Tick` objects into `RenkoBar` objects. Follow these steps to create and manage a `Tick` consolidator based on the preceding Renko bar rules:

1. Create the consolidator.

   To create a classic Renko consolidator, pass the bar size to the `ClassicRenkoConsolidator` constructor.

   ```C#
   // Create a Classic Renko consolidator that emits a bar when the price moves $1
   _consolidator = new ClassicRenkoConsolidator(1m);
   ```

   The `ClassicRenkoConsolidator` has the following default behavior:

   - It uses the `Value` property of the `IBaseData` object it receives to build the Renko bars
   - It ignores the volume of the input data
   - It enforces the open and close of each bar to be a multiple of the bar size

   To relax the requirement that the open and close of the Renko bars must be a multiple of bar size, disable the `evenBars` argument. If you disable `evenBars` , the open value of the first Renko bar is set to the first value from the `selector` . The following opening and closing Renko bar values are all multiples of the first value from the `selector`

   ```C#
   _consolidator = new ClassicRenkoConsolidator(1m, evenBars: false);
   ```

2. Add an event handler to the consolidator.

   ```C#
   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

```
C#
void ConsolidationHandler(object sender, RenkoBar consolidatedBar)
{

}
```

The consolidation event handler receives bars when the price movement forms a new classic Renko bar.

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
C#
SubscriptionManager.AddConsolidator(_symbol, _consolidator);
```

- Manual Updates

Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
C#
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var ticks = slice.Ticks[_symbol];
    foreach (var tick in ticks)
    {
     _consolidator.Update(tick);
    }
}

// Example 2: Update the consolidator with data from a history request
var ticks = History<Tick>(_symbol, TimeSpan.FromMinutes(3), Resolution.Tick);
foreach (var tick in ticks)
{
    _consolidator.Update(tick);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Quote Ticks

`Tick` quote bar consolidators aggregate `Tick` objects that represent quotes into `RenkoBar` objects. Follow these steps to create and manage a `Tick` quote bar consolidator based on the preceding Renko bar rules:

1. Create the consolidator.

   To create a classic Renko consolidator, pass the bar size to the `ClassicRenkoConsolidator` constructor.

   ```
   // Create a Classic Renko consolidator that emits a bar when the price moves $1
   _consolidator = new ClassicRenkoConsolidator(1m);
   ```

   The `ClassicRenkoConsolidator` has the following default behavior:

   - It uses the `Value` property of the `IBaseData` object it receives to build the Renko bars
   - It ignores the volume of the input data
   - It enforces the open and close of each bar to be a multiple of the bar size

   To relax the requirement that the open and close of the Renko bars must be a multiple of bar size, disable the `evenBars` argument. If you disable `evenBars` , the open value of the first Renko bar is set to the first value from the `selector` . The following opening and closing Renko bar values are all multiples of the first value from the `selector`

   ```
   _consolidator = new ClassicRenkoConsolidator(1m, evenBars: false);
   ```

2. Add an event handler to the consolidator.

   ```
   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

   ```
   void ConsolidationHandler(object sender, RenkoBar consolidatedBar)
   {

   }
   ```

The consolidation event handler receives bars when the price movement forms a new classic Renko bar.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

   - Automatic Updates

     To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

     ```
     C#

     SubscriptionManager.AddConsolidator(_symbol, _consolidator);
     ```

   - Manual Updates

     Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

     ```
     C#

     // Example 1: Update the consolidator with data from the Slice object
     public override void OnData(Slice slice)
     {
         var ticks = slice.Ticks[_symbol];
         foreach (var tick in ticks)
         {
          _consolidator.Update(tick);
         }
     }

     // Example 2: Update the consolidator with data from a history request
     var ticks = History<Tick>(_symbol, TimeSpan.FromMinutes(3), Resolution.Tick);
     foreach (var tick in ticks)
     {
         _consolidator.Update(tick);
     }
     ```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   C#

   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Reset Consolidators

To reset a consolidator, call its `Reset` method.

```
C#

_consolidator.Reset();
```

If you are live trading Equities or backtesting Equities without the adjusted data normalization mode , reset your consolidators

when splits and dividends occur. When a split or dividend occurs while the consolidator is in the process of building a bar, the open, high, and low may reflect prices from before the split or dividend. To avoid issues, call the consolidator's `Reset` method and then warm it up with `ScaledRaw` data from a history request .

```
C#
public override void OnData(Slice data)
{
    // When a split or dividend occurs...
    if ((data.Splits.ContainsKey(_symbol) && data.Splits[_symbol].Type == SplitType.SplitOccurred) ||
        data.Dividends.ContainsKey(_symbol))
    {
        // If the consolidator is working on a bar...
        if (_consolidator.WorkingData != null)
        {
            // Get adjusted prices for the time period of the working bar.
            var history = History<TradeBar>(_symbol, _consolidator.WorkingData.Time, Time, dataNormalizationMode:
DataNormalizationMode.ScaledRaw);
            // Reset the consolidator.
            _consolidator.Reset();
            // Warm-up the consolidator with the adjusted price data.
            foreach (var bar in history)
            {
                _consolidator.Update(bar);
            }
        }
    }
}
```

Examples

The following examples demonstrate some common practices for implementing classic renko consolidator.

Example 1: 10 Dollar Bar Range SMA Cross

The following algorithm creates a dollar bar for SPY. Then, for each 10-dollar bar, we find their range (high - low) and feed to an SMA indicator, such that we obtain a trend indicator of the range of SPY. We trade VXX as the proxy of the SPY volatility based on the trend of the range SMA.

C#

```
public class ClassicRenkoBarConsolidatorAlgorithm : QCAlgorithm
{
    private Symbol _spy, _vxx;
    private ClassicRenkoConsolidator _consolidator;
    private SimpleMovingAverage _sma = new(10);
    private decimal _high = 0m, _low = 10000m;
    private int _count = 0;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request SPY data to feed the consolidators and indicators for trade signal generation.
        _spy = AddEquity("SPY", Resolution.Minute).Symbol;
        // Request VXX data for trading.
        _vxx = AddEquity("VXX", Resolution.Minute).Symbol;

        // Create a dollar bar of SPY to track the volatility.
        _consolidator = new ClassicRenkoConsolidator(1);
        _consolidator.DataConsolidated += OnConsolidated;

        // Subscribe the consolidators to SPY data for automatic updating.
        SubscriptionManager.AddConsolidator(_spy, _consolidator);

        _sma.Window.Size = 2;
        SetWarmUp(100);
    }

    private void OnConsolidated(object sender, TradeBar bar)
    {
        // Update the SPY range of the current batch.
        _high = Math.Max(_high, bar.High);
        _low = Math.Min(_low, bar.Low);
        _count++;

        // Update SMA of SPY range of the last 10 Renko bars.
        if (_count >= 10)
        {
            _sma.Update(bar.EndTime, _high - _low);
            // Reset high and low.
            _high = 0m;
            _low = 10000m;
            _count = 0;
        }
    }

    public override void OnData(Slice slice)
    {
        if (_sma.IsReady)
        {
            // If the range trend is increasing, buy VXX as a proxy for the increase in volatility.
            if (_sma > _sma.Previous && !Portfolio[_vxx].IsLong)
            {
                SetHoldings(_vxx, 0.5m);
            }
            // If the range trend is decreasing, sell VXX as a proxy for decreasing volatility.
            else if (_sma < _sma.Previous && !Portfolio[_vxx].IsShort)
            {
                SetHoldings(_vxx, -0.5m);
            }
        }
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
ClassicRenkoConsolidatorAlgorithm.cs C#

Renko Consolidators

Volume Renko Consolidators

Introduction

Volume Renko consolidators aggregate bars based on a fixed trading volume. These types of bars are commonly known as volume bars. A `VolumeRenkoConsolidator` with a bar size of 10,000 produces a new bar every time 10,000 units of the security trades in the market. If the trading volume of a single time step exceeds two step sizes (i.e. >20,000), the `VolumeRenkoConsolidator` produces multiple bars in a single time step.

Consolidate Trade Bars

`TradeBar` consolidators aggregate `TradeBar` objects into `VolumeRenkoBar` objects. Follow these steps to create and manage a `TradeBar` consolidator based on custom set volume traded:

1. Create the consolidator.

   To create a Volume Renko consolidator, pass the bar size to the `VolumeRenkoConsolidator` constructor.

   ```
   C#
   // Create a Volume Renko consolidator that emits a bar every time 10,000 units trade
   _consolidator = new VolumeRenkoConsolidator(10000m);
   ```

2. Add an event handler to the consolidator.

   ```
   C#
   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

   ```
   C#
   void ConsolidationHandler(object sender, VolumeRenkoBar consolidatedBar)
   {

   }
   ```

   The consolidation event handler receives bars when the trading volume forms a new Volume Renko bar.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

   - Automatic Updates

To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
SubscriptionManager.AddConsolidator(_symbol, _consolidator);
```

- Manual Updates

Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `TradeBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var tradeBar = slice.Bars[_symbol];
    _consolidator.Update(tradeBar);
}

// Example 2: Update the consolidator with data from a history request
var history = History<TradeBar>(_symbol, 30, Resolution.Minute);
foreach (var tradeBar in history)
{
    _consolidator.Update(tradeBar);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Trade Ticks

`Tick` consolidators aggregate `Tick` objects into `VolumeRenkoBar` objects. Follow these steps to create and manage a `Tick` consolidator based on custom set volume traded:

1. Create the consolidator.

To create a Volume Renko consolidator, pass the bar size to the `VolumeRenkoConsolidator` constructor.

```
// Create a Volume Renko consolidator that emits a bar every time 10,000 units trade
_consolidator = new VolumeRenkoConsolidator(10000m);
```

2. Add an event handler to the consolidator.

```
_consolidator.DataConsolidated += ConsolidationHandler;
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler`, not `ConsolidationHandler()`.

3. Define the consolidation handler.

```
C#
void ConsolidationHandler(object sender, VolumeRenkoBar consolidatedBar)
{

}
```

The consolidation event handler receives bars when the trading volume forms a new Volume Renko bar.

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
C#
SubscriptionManager.AddConsolidator(_symbol, _consolidator);
```

- Manual Updates

Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period, you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request.

```
C#
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var ticks = slice.Ticks[_symbol];
    foreach (var tick in ticks)
    {
     _consolidator.Update(tick);
    }
}

// Example 2: Update the consolidator with data from a history request
var ticks = History<Tick>(_symbol, TimeSpan.FromMinutes(3), Resolution.Tick);
foreach (var tick in ticks)
{
    _consolidator.Update(tick);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Reset Consolidators

To reset a consolidator, call its `Reset` method.

```
_consolidator.Reset();
```

If you are live trading Equities or backtesting Equities without the adjusted data normalization mode , reset your consolidators when splits and dividends occur. When a split or dividend occurs while the consolidator is in the process of building a bar, the open, high, and low may reflect prices from before the split or dividend. To avoid issues, call the consolidator's `Reset` method and then warm it up with `ScaledRaw` data from a history request .

```
public override void OnData(Slice data)
{
    // When a split or dividend occurs...
    if ((data.Splits.ContainsKey(_symbol) && data.Splits[_symbol].Type == SplitType.SplitOccurred) ||
        data.Dividends.ContainsKey(_symbol))
    {
        // If the consolidator is working on a bar...
        if (_consolidator.WorkingData != null)
        {
            // Get adjusted prices for the time period of the working bar.
            var history = History<TradeBar>(_symbol, _consolidator.WorkingData.Time, Time, dataNormalizationMode:
DataNormalizationMode.ScaledRaw);
            // Reset the consolidator.
            _consolidator.Reset();
            // Warm-up the consolidator with the adjusted price data.
            foreach (var bar in history)
            {
                _consolidator.Update(bar);
            }
        }
    }
}
```

Examples

The following examples demonstrate common practices for implementing volume renko consolidator.

Example 1: Bollinger Bands From Volume Renko Bar

The following algorithm uses 100,000-share volume Renko bars to spot mean reversal opportunities on SPY with Bollinger Bands. Equal volume bars could provide more equal information in each bar, so the standard deviation and trade opportunities are more credible.

C#

```
public class VolumeRenkoBarConsolidatorAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private VolumeRenkoConsolidator _consolidator;
    // We use a Bollinger band to signal mean reversal opportunities.
    private BollingerBands _bbands = new(20, 2, MovingAverageType.Exponential);

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request SPY data to feed for the consolidators, indicators, and trade signal generation.
        _spy = AddEquity("SPY", Resolution.Minute).Symbol;

        // Create a 100,000-share bar of SPY to obtain a more equally-informed bar.
        _consolidator = new VolumeRenkoConsolidator(100000m);

        // Subscribe the consolidators to SPY data to automatically update the indicator.
        RegisterIndicator(_spy, _bbands, _consolidator);

        // Warm up the indicator for immediate use.
        SetWarmUp(20);
    }

    public override void OnData(Slice slice)
    {
        if (!IsWarmingUp && _bbands.IsReady && slice.Bars.TryGetValue(_spy, out var bar))
        {
            // Trade the mean reversal by short-selling SPY if the current price exceeds the upper threshold.
            if (bar.Close > _bbands.UpperBand && !Portfolio[_spy].IsShort)
            {
                SetHoldings(_spy, -0.5m);
            }
            // Trade the mean reversal by buying SPY if the current price is below the lower threshold.
            else if (bar.Close < _bbands.LowerBand && !Portfolio[_spy].IsLong)
            {
                SetHoldings(_spy, 0.5m);
            }
            // Exit position if the price is mean-reverted that crosses the EMA reference.
            else if ((Portfolio[_spy].Quantity > 0 && bar.Close > _bbands.MiddleBand) ||
                (Portfolio[_spy].Quantity < 0 && bar.Close < _bbands.MiddleBand))
            {
                Liquidate(_spy);
            }
        }
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
VolumeRenkoConsolidatorAlgorithm.cs C#

Consolidator Types

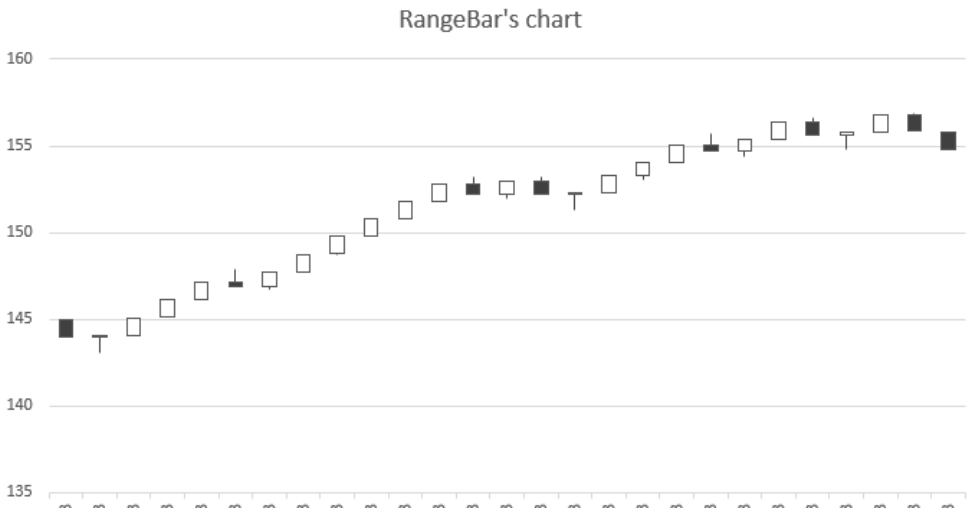
Range Consolidators

Range Consolidators

Range Consolidators

Introduction

Most Range consolidators aggregate bars based on a fixed price movement. It is very much like a Renko Bar, but instead of consolidating a fix movement from the opening price, a Range Bar would fix the range of the whole bar, i.e. High - Low. A new bar is formed when the range of the whole bar reached a preset value, thus it completely neglects the time effect and focuses on the variance/information carried by the price movement.



Consolidate Trade Bars

`TradeBar` consolidators aggregate `TradeBar` objects into `RangeBar` objects. Follow these steps to create and manage a `TradeBar` consolidator based on the preset range of price movement:

1. Create the consolidator.

   To create a Range consolidator, pass the bar size to the `RangeConsolidator` constructor.

   ```
   // Create a Range consolidator that emits a bar when the high and low differs $1
   _consolidator = new RangeConsolidator(1m);
   ```

2. Add an event handler to the consolidator.

   ```
   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

```
C#
void ConsolidationHandler(object sender, RangeBar consolidatedBar)
{

}
```

The consolidation event handler receives bars when the price movement forms a new Range bar.

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
C#
SubscriptionManager.AddConsolidator(_symbol, _consolidator);
```

- Manual Updates

Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `TradeBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
C#
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var tradeBar = slice.Bars[_symbol];
    _consolidator.Update(tradeBar);
}

// Example 2: Update the consolidator with data from a history request
var history = History<TradeBar>(_symbol, 30, Resolution.Minute);
foreach (var tradeBar in history)
{
    _consolidator.Update(tradeBar);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
C#
SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow

down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Quote Bars

`QuoteBar` consolidators aggregate `QuoteBar` objects into `RangeBar` objects. Follow these steps to create and manage a `QuoteBar` consolidator based on the preset range of price movement:

1. Create the consolidator.

   To create a Range consolidator, pass the bar size to the `RangeConsolidator` constructor.

   ```
   C#

   // Create a Range consolidator that emits a bar when the high and low differs $1
   _consolidator = new RangeConsolidator(1m);
   ```

2. Add an event handler to the consolidator.

   ```
   C#

   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

   ```
   C#

   void ConsolidationHandler(object sender, RangeBar consolidatedBar)
   {

   }
   ```

   The consolidation event handler receives bars when the price movement forms a new Range bar.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

   - Automatic Updates

     To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

     ```
     C#

     SubscriptionManager.AddConsolidator(_symbol, _consolidator);
     ```

   - Manual Updates

     Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `QuoteBar` object. You can update the consolidator with

data from the Slice object in the `OnData` method or with data from a history request .

```
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var quoteBar = slice.QuoteBars[_symbol];
    _consolidator.Update(quoteBar);
}

// Example 2: Update the consolidator with data from a history request
var history = History<QuoteBar>(_symbol, 30, Resolution.Minute);
foreach (var quoteBar in history)
{
    _consolidator.Update(quoteBar);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Trade Ticks

`Tick` consolidators aggregate `Tick` objects into `RangeBar` objects. Follow these steps to create and manage a `Tick` consolidator based on the preset range of price movement:

1. Create the consolidator.

   To create a Range consolidator, pass the bar size to the `RangeConsolidator` constructor.

```
// Create a Range consolidator that emits a bar when the high and low differs $1
_consolidator = new RangeConsolidator(1m);
```

2. Add an event handler to the consolidator.

```
_consolidator.DataConsolidated += ConsolidationHandler;
```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

```
void ConsolidationHandler(object sender, RangeBar consolidatedBar)
{

}
```

The consolidation event handler receives bars when the price movement forms a new Range bar.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

   - Automatic Updates

     To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

     ```
     SubscriptionManager.AddConsolidator(_symbol, _consolidator);
     ```

   - Manual Updates

     Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

     ```
     // Example 1: Update the consolidator with data from the Slice object
     public override void OnData(Slice slice)
     {
         var ticks = slice.Ticks[_symbol];
         foreach (var tick in ticks)
         {
          _consolidator.Update(tick);
         }
     }

     // Example 2: Update the consolidator with data from a history request
     var ticks = History<Tick>(_symbol, TimeSpan.FromMinutes(3), Resolution.Tick);
     foreach (var tick in ticks)
     {
         _consolidator.Update(tick);
     }
     ```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Quote Ticks

`Tick` quote bar consolidators aggregate `Tick` objects that represent quotes into `RangeBar` objects. Follow these steps to create

and manage a `Tick` quote bar consolidator based on the preset range of price movement:

1. Create the consolidator.

   To create a Range consolidator, pass the bar size to the `RangeConsolidator` constructor.

   ```
   C#
   // Create a Range consolidator that emits a bar when the high and low differs $1
   _consolidator = new RangeConsolidator(1m);
   ```

2. Add an event handler to the consolidator.

   ```
   C#
   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

   ```
   C#
   void ConsolidationHandler(object sender, RangeBar consolidatedBar)
   {

   }
   ```

   The consolidation event handler receives bars when the price movement forms a new Range bar.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

   - Automatic Updates

     To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

     ```
     C#
     SubscriptionManager.AddConsolidator(_symbol, _consolidator);
     ```

   - Manual Updates

     Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```C#
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var ticks = slice.Ticks[_symbol];
    foreach (var tick in ticks)
    {
     _consolidator.Update(tick);
    }
}

// Example 2: Update the consolidator with data from a history request
var ticks = History<Tick>(_symbol, TimeSpan.FromMinutes(3), Resolution.Tick);
foreach (var tick in ticks)
{
    _consolidator.Update(tick);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```C#
   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Reset Consolidators

To reset a consolidator, call its `Reset` method.

```C#
_consolidator.Reset();
```

If you are live trading Equities or backtesting Equities without the adjusted data normalization mode , reset your consolidators when splits and dividends occur. When a split or dividend occurs while the consolidator is in the process of building a bar, the open, high, and low may reflect prices from before the split or dividend. To avoid issues, call the consolidator's `Reset` method and then warm it up with `ScaledRaw` data from a history request .

```C#
public override void OnData(Slice data)
{
    // When a split or dividend occurs...
    if ((data.Splits.ContainsKey(_symbol) && data.Splits[_symbol].Type == SplitType.SplitOccurred) ||
        data.Dividends.ContainsKey(_symbol))
    {
        // If the consolidator is working on a bar...
        if (_consolidator.WorkingData != null)
        {
            // Get adjusted prices for the time period of the working bar.
            var history = History<TradeBar>(_symbol, _consolidator.WorkingData.Time, Time, dataNormalizationMode:
DataNormalizationMode.ScaledRaw);
            // Reset the consolidator.
            _consolidator.Reset();
            // Warm-up the consolidator with the adjusted price data.
            foreach (var bar in history)
            {
                _consolidator.Update(bar);
            }
        }
    }
}
```

Range Consolidators

Classic Range Consolidators

Introduction

Most Range consolidators aggregate bars based on a fixed price range movement. The `ClassicRangeConsolidator` produces a different type of Range bars than the `RangeConsolidator` . A `ClassicRangeConsolidator` with a bar size of 1*producesanewbarthatspans*1 every time an asset's price range moves $1. If the price jumps multiple dollars in a single tick, the `ClassicRangeConsolidator` only produces one bar per time step where the open of each bar matches the close of the previous bar.

Consolidate Trade Bars

`TradeBar` consolidators aggregate `TradeBar` objects into `RangeBar` objects. Follow these steps to create and manage a `TradeBar` consolidator based on the preceding Range bar rules:

1. Create the consolidator.

   To create a classic Range consolidator, pass the bar size to the `ClassicRangeConsolidator` constructor.

   ```
   C#
   // Create a Classic Range consolidator that emits a bar when the price range moves $1
   _consolidator = new ClassicRangeConsolidator(1m);
   ```

   The `ClassicRangeConsolidator` has the following default behavior:

   - It uses the `Value` property of the `IBaseData` object it receives to build the Range bars
   - It ignores the volume of the input data
   - It enforces the high and low of each bar to be a multiple of the bar size

   To build the Range bars with a different property than the `Value` of the `IBaseData` object, provide a `selector` argument. The `selector` should be a function that receives the `IBaseData` object and returns a decimal value.

   ```
   C#
   _consolidator = new ClassicRangeConsolidator(1m, data => (data as TradeBar).High);
   ```

   To add a non-zero `Volume` property to the Range bars, provide a `volumeSelector` argument. The `volumeSelector` should be a function that receives the `IBaseData` object and returns a decimal value.

   ```
   C#
   _consolidator = new ClassicRangeConsolidator(1m, null, data => (data as TradeBar).Volume);
   ```

2. Add an event handler to the consolidator.

   ```
   C#
   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating

consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

```
C#
void ConsolidationHandler(object sender, RangeBar consolidatedBar)
{

}
```

The consolidation event handler receives bars when the price movement forms a new classic Range bar.

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
C#
SubscriptionManager.AddConsolidator(_symbol, _consolidator);
```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `TradeBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
C#
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var tradeBar = slice.Bars[_symbol];
    _consolidator.Update(tradeBar);
}

// Example 2: Update the consolidator with data from a history request
var history = History<TradeBar>(_symbol, 30, Resolution.Minute);
foreach (var tradeBar in history)
{
    _consolidator.Update(tradeBar);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
C#
SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions,

see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Quote Bars

`QuoteBar` consolidators aggregate `QuoteBar` objects into `RangeBar` objects. Follow these steps to create and manage a `QuoteBar` consolidator based on the preceding Range bar rules:

1. Create the consolidator.

   To create a classic Range consolidator, pass the bar size to the `ClassicRangeConsolidator` constructor.

   ```
   C#

   // Create a Classic Range consolidator that emits a bar when the price range moves $1
   _consolidator = new ClassicRangeConsolidator(1m);
   ```

   The `ClassicRangeConsolidator` has the following default behavior:

   - It uses the `Value` property of the `IBaseData` object it receives to build the Range bars
   - It ignores the volume of the input data
   - It enforces the high and low of each bar to be a multiple of the bar size

   The following arguments enable you to create Range bars that aggregate the excessive bid-ask spread on the asset. It can help market-making algorithms to track wider spread opportunities.

   ```
   C#

   _consolidator = new ClassicRangeConsolidator(10, null,
       data => (data as QuoteBar).LastBidSize - (data as QuoteBar).LastAskSize);
   ```

2. Add an event handler to the consolidator.

   ```
   C#

   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

   ```
   C#

   void ConsolidationHandler(object sender, RangeBar consolidatedBar)
   {

   }
   ```

   The consolidation event handler receives bars when the price movement forms a new classic Range bar.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

   - Automatic Updates

To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

```
C#
SubscriptionManager.AddConsolidator(_symbol, _consolidator);
```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `QuoteBar` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

```
C#
// Example 1: Update the consolidator with data from the Slice object
public override void OnData(Slice slice)
{
    var quoteBar = slice.QuoteBars[_symbol];
    _consolidator.Update(quoteBar);
}

// Example 2: Update the consolidator with data from a history request
var history = History<QuoteBar>(_symbol, 30, Resolution.Minute);
foreach (var quoteBar in history)
{
    _consolidator.Update(quoteBar);
}
```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
C#
SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Trade Ticks

`Tick` consolidators aggregate `Tick` objects into `RangeBar` objects. Follow these steps to create and manage a `Tick` consolidator based on the preceding Range bar rules:

1. Create the consolidator.

   To create a classic Range consolidator, pass the bar size to the `ClassicRangeConsolidator` constructor.

```
C#
// Create a Classic Range consolidator that emits a bar when the price range moves $1
_consolidator = new ClassicRangeConsolidator(1m);
```

   The `ClassicRangeConsolidator` has the following default behavior:

   - It uses the `Value` property of the `IBaseData` object it receives to build the Range bars
   - It ignores the volume of the input data
   - It enforces the high and low of each bar to be a multiple of the bar size

2. Add an event handler to the consolidator.

```
C#
_consolidator.DataConsolidated += ConsolidationHandler;
```

LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

```
C#
void ConsolidationHandler(object sender, RangeBar consolidatedBar)
{

}
```

The consolidation event handler receives bars when the price movement forms a new classic Range bar.

4. Update the consolidator.

You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  C#
  SubscriptionManager.AddConsolidator(_symbol, _consolidator);
  ```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

  ```
  C#
  // Example 1: Update the consolidator with data from the Slice object
  public override void OnData(Slice slice)
  {
      var ticks = slice.Ticks[_symbol];
      foreach (var tick in ticks)
      {
       _consolidator.Update(tick);
      }
  }

  // Example 2: Update the consolidator with data from a history request
  var ticks = History<Tick>(_symbol, TimeSpan.FromMinutes(3), Resolution.Tick);
  foreach (var tick in ticks)
  {
      _consolidator.Update(tick);
  }
  ```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

```
C#
SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
```

If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Consolidate Quote Ticks

`Tick` quote bar consolidators aggregate `Tick` objects that represent quotes into `RangeBar` objects. Follow these steps to create and manage a `Tick` quote bar consolidator based on the preceding Range bar rules:

1. Create the consolidator.

   To create a classic Range consolidator, pass the bar size to the `ClassicRangeConsolidator` constructor.

   ```
   C#
   // Create a Classic Range consolidator that emits a bar when the price range moves $1
   _consolidator = new ClassicRangeConsolidator(1m);
   ```

   The `ClassicRangeConsolidator` has the following default behavior:

   - It uses the `Value` property of the `IBaseData` object it receives to build the Range bars
   - It ignores the volume of the input data
   - It enforces the high and low of each bar to be a multiple of the bar size

2. Add an event handler to the consolidator.

   ```
   C#
   _consolidator.DataConsolidated += ConsolidationHandler;
   ```

   LEAN passes consolidated bars to the consolidator event handler in your algorithm. The most common error when creating consolidators is to put parenthesis `()` at the end of your method name when setting the event handler of the consolidator. If you use parenthesis, the method executes and the result is passed as the event handler instead of the method itself. Remember to pass the name of your method to the event system. Specifically, it should be `ConsolidationHandler` , not `ConsolidationHandler()` .

3. Define the consolidation handler.

   ```
   C#
   void ConsolidationHandler(object sender, RangeBar consolidatedBar)
   {

   }
   ```

   The consolidation event handler receives bars when the price movement forms a new classic Range bar.

4. Update the consolidator.

   You can automatically or manually update the consolidator.

- Automatic Updates

  To automatically update a consolidator with data from the security subscription, call the `AddConsolidator` method of the Subscription Manager.

  ```
  C#
  SubscriptionManager.AddConsolidator(_symbol, _consolidator);
  ```

- Manual Updates

  Manual updates let you control when the consolidator updates and what data you use to update it. If you need to warm up a consolidator with data outside of the warm-up period , you can manually update the consolidator. To manually update a consolidator, call its `Update` method with a `Tick` object. You can update the consolidator with data from the Slice object in the `OnData` method or with data from a history request .

  ```
  C#
  // Example 1: Update the consolidator with data from the Slice object
  public override void OnData(Slice slice)
  {
      var ticks = slice.Ticks[_symbol];
      foreach (var tick in ticks)
      {
       _consolidator.Update(tick);
      }
  }

  // Example 2: Update the consolidator with data from a history request
  var ticks = History<Tick>(_symbol, TimeSpan.FromMinutes(3), Resolution.Tick);
  foreach (var tick in ticks)
  {
      _consolidator.Update(tick);
  }
  ```

5. If you create consolidators for securities in a dynamic universe and register them for automatic updates, remove the consolidator when the security leaves the universe.

   ```
   C#
   SubscriptionManager.RemoveConsolidator(_symbol, _consolidator);
   ```

   If you have a dynamic universe and don't remove consolidators, they compound internally, causing your algorithm to slow down and eventually die once it runs out of RAM. For an example of removing consolidators from universe subscriptions, see the GasAndCrudeOilEnergyCorrelationAlpha in the LEAN GitHub repository.

Reset Consolidators

To reset a consolidator, call its `Reset` method.

```
C#
_consolidator.Reset();
```

If you are live trading Equities or backtesting Equities without the adjusted data normalization mode , reset your consolidators when splits and dividends occur. When a split or dividend occurs while the consolidator is in the process of building a bar, the open, high, and low may reflect prices from before the split or dividend. To avoid issues, call the consolidator's `Reset` method and then warm it up with `ScaledRaw` data from a history request .

```
C#
public override void OnData(Slice data)
{
    // When a split or dividend occurs...
    if ((data.Splits.ContainsKey(_symbol) && data.Splits[_symbol].Type == SplitType.SplitOccurred) ||
        data.Dividends.ContainsKey(_symbol))
    {
        // If the consolidator is working on a bar...
        if (_consolidator.WorkingData != null)
        {
            // Get adjusted prices for the time period of the working bar.
            var history = History<TradeBar>(_symbol, _consolidator.WorkingData.Time, Time, dataNormalizationMode:
DataNormalizationMode.ScaledRaw);
            // Reset the consolidator.
            _consolidator.Reset();
            // Warm-up the consolidator with the adjusted price data.
            foreach (var bar in history)
            {
                _consolidator.Update(bar);
            }
        }
    }
}
```

Examples

Demonstration Algorithms
ClassicRangeConsolidatorAlgorithm.cs C#

Consolidator Types

Combining Consolidators

Introduction

Sequential consolidators wire two internal consolidators together such that the output of the first consolidator is the input to the second consolidator and the output of the second consolidator is the output of the sequential consolidator.

Create Consolidators

To create a sequential consolidator, create two consolidators and then pass them to the `SequentialConsolidator` constructor.

```C#
// This first consolidator produces a consolidated bar after a day passes
var oneDayConsolidator = new TradeBarConsolidator(TimeSpan.FromDays(1));

// This second consolidators produces a consolidated bar after it sees 3 samples
var threeCountConsolidator = new TradeBarConsolidator(3);

// This sequential consolidator aggregates three 1-day bars together
_consolidator = new SequentialConsolidator(oneDayConsolidator, threeCountConsolidator);
```

For more information about each type of consolidator, see Consolidator Types .

Examples

Demonstration Algorithms
DataConsolidationAlgorithm.cs C#

Consolidating Data

Consolidator History

Introduction

This page explains how to save and access historical consolidated bars.

Save Consolidated Bars

To access historical bars that were passed to your consolidation handler, save the bars as you receive them. You can use a RollingWindow to save the consolidated bars and easily access them later on in your algorithm.

```C#
// Create a class member to store the RollingWindow
_window = new RollingWindow<TradeBar>(2);

// In the consolidation handler, add consolidated bars to the RollingWindow
private void ConsolidationHandler(object sender, TradeBar consolidatedBar)
{
    _window.Add(consolidatedBar);
}
```

Get Historical Bars

If you save consolidated bars in a `RollingWindow`, you can access them by indexing the `RollingWindow`. `RollingWindow` objects operate on a first-in, first-out process to allow for reverse list access semantics with positive and negative indexing. Index 0 refers to the most recent item in the window and the largest index refers to the last item in the window. Index -1 refers to the latest item as well.

```C#
var mostRecentBar = _window[0];
var previousBar = _window[1];
var oldestBar = _window[-1];
```

To get the consolidated bar that was most recently removed from the `RollingWindow`, use the `MostRecentlyRemoved` property.

```C#
var removedBar = _window.MostRecentlyRemoved;
```

Examples

The following examples demonstrate some common practices for consolidator history.

Example 1: Price Action

The following algorithm trades breakout price action on the SPY five-minute trade bar. To do so, we must create a five-minute trade bar consolidator and a rolling window to hold 3 trade bars to check if the trade conditions are fulfilled.

C#

```
public class ConsolidatorHistoryAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    // To hold 3 consolidated trade bars to identify a breakout pattern.
    private RollingWindow<TradeBar> _windows = new(3);

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request SPY data for signal generation and trading.
        _spy = AddEquity("SPY", Resolution.Minute).Symbol;

        // The breakout is based on a 5-minute consolidated trade bar.
        var consolidator = new TradeBarConsolidator(TimeSpan.FromMinutes(5));
        // Subscribe for automatically updating the consolidator with SPY data.
        SubscriptionManager.AddConsolidator(_spy, consolidator);
        // Add a consolidator handler to check that the breakout condition is fulfilled and traded.
        consolidator.DataConsolidated += OnConsolidated;

        SetWarmUp(TimeSpan.FromDays(1));
    }

    private void OnConsolidated(object sender, TradeBar bar)
    {
        _windows.Add(bar);

        if (!IsWarmingUp && _windows.IsReady)
        {
            // Buy if the breakout price action is fulfilled.
            // 1. Increasing price trend.
            // 2. The last 3 bars are green.
            // 3. The 3rd and 2nd last bars range is decreasing.
            // 4. The last bar exceeds the 2nd last bar by double the 2nd last bar's range.
            var secondLastRange = _windows[1].Close - _windows[1].Open;
            if (bar.Close > _windows[1].Close && _windows[1].Close > _windows[2].Close &&
            _windows[2].Close > _windows[2].Open && _windows[1].Close > _windows[1].Open && _windows[0].Close >
_windows[0].Open &&
            _windows[2].Close - _windows[2].Open > secondLastRange && bar.Close > _windows[1].Close + 2 *
secondLastRange)
            {
                SetHoldings(_spy, 0.5m);
            }
        }
    }

    public override void OnOrderEvent(OrderEvent orderEvent)
    {
        if (orderEvent.Status == OrderStatus.Filled)
        {
            if (orderEvent.Ticket.OrderType == OrderType.Market)
            {
                // Stop loss order at 1%.
                var stopPrice = orderEvent.FillQuantity > 0m ? orderEvent.FillPrice * 0.99m : orderEvent.FillPrice *
1.01m;
                StopMarketOrder(_spy, -Portfolio[_spy].Quantity, stopPrice);
                // Take profit order at 2%.
                var takeProfitPrice = orderEvent.FillQuantity > 0m ? orderEvent.FillPrice * 1.02m :
orderEvent.FillPrice * 0.98m;
                LimitOrder(_spy, -Portfolio[_spy].Quantity, takeProfitPrice);
            }
            else if (orderEvent.Ticket.OrderType == OrderType.StopMarket || orderEvent.Ticket.OrderType ==
OrderType.Limit)
            {
                // Cancel any open order if stop loss or take profit order filled.
                Transactions.CancelOpenOrders();
            }
        }
    }
}
```

Consolidating Data

Updating Indicators

Introduction

You can use consolidators to automatically update indicators in your algorithms. The consolidators can update your indicators at each time step or with aggregated bars. By default, LEAN updates data point indicators with the close price of the consolidated bars, but you can change it to a custom data field.

Standard Indicator Periods

If your algorithm has a static universe, you can create automatic indicators in just one line of code. When you create an automatic indicator, LEAN creates a consolidator, hooks it up for automatic updates, and then updates the indicator with the consolidated bars.

```C#
// Consolidate minute SPY data into 14-bar daily indicators
var ema = EMA("SPY", 14, Resolution.Daily);
var sma = SMA("SPY", 14, Resolution.Daily);
```

If your algorithm has a dynamic universe, create a manual indicator and then create a time period consolidator that updates the indicator at each time step. Keep a reference to the consolidator so you can remove it when your algorithm removes the security from the universe.

```C#
public override void Initialize()
{
    _ema = new ExponentialMovingAverage(14);
    _consolidator = new TradeBarConsolidator(1);
    _consolidator.DataConsolidated += OnDataConsolidated;
    SubscriptionManager.AddConsolidator(_symbol, _consolidator);
}

// Define consolidator handler function as a class method
public void OnDataConsolidated(object sender, TradeBar bar)
{
    _ema.Update(bar.EndTime, bar.Close);
}
```

Custom Indicator Periods

It's common to update indicators with price data that spans a normal time period like one minute or one day. It's less common to update an indicator with exotic time periods (for example, a 7-minute consolidated bar), so these types of indicators may provide more opportunities for alpha. To update indicators with exotic data, create a manual indicator and then call the `RegisterIndicator` method. The `RegisterIndicator` method wires up the indicator for automatic updates at the time interval you provide.

```C#
// Calculate the SMA with 10 7-minute bars
var symbol = AddEquity("SPY", Resolution.Minute).Symbol;
var indicator = new SimpleMovingAverage(10);
RegisterIndicator(symbol, indicator, TimeSpan.FromMinutes(7));
```

The RegisterIndicator method can accept a TimeSpan , Resolution , or an unregistered consolidator. If you apply the indicator to a security in a dynamic universe, provide a consolidator so that you can remove it when your algorithm removes the security from the universe.

```C#
// TimeSpan
RegisterIndicator(symbol, indicator, TimeSpan.FromMinutes(7));

// Resolution
RegisterIndicator(symbol, indicator, Resolution.Hour);

// Consolidator
_consolidator = new TradeBarConsolidator(35);
RegisterIndicator(symbol, indicator, _consolidator);
```

Custom Indicator Values

Data point indicators use only a single price data in their calculations. By default, those indicators use the closing price. For assets with TradeBar data, that price is the TradeBar close price. For assets with QuoteBar data, that price is the mid-price of the bid closing price and the ask closing price. To create an indicator with the other fields like the Open , High , Low , or Close , provide a selector argument to the RegisterIndicator method.

```C#
// Define a 10-period RSI with indicator the constructor
_rsi = new RelativeStrengthIndex(10, MovingAverageType.Simple);

// Register the daily High price data to automatically update the indicator
RegisterIndicator(symbol, _rsi, Resolution.Daily, Field.High);
```

The RegisterIndicator method can accept a TimeSpan , Resolution , or an unregistered consolidator. If you apply the indicator to a security in a dynamic universe, provide a consolidator so that you can remove it when your algorithm removes the security from the universe.

The Field class has the following selector properties:

To create a custom selector , define a function that calculates the value.

```C#
RegisterIndicator(_symbol, _rsi,  Resolution.Daily, x =>
{
    var bar = x as IBaseDataBar;
    return (bar.Low + bar.High) / 2;
});
```

Examples

The following examples demonstrate common practices for updating indicators using consolidators.

Example 1: Hammer Pattern With 5-Minute Consolidator

The following algorithm trades Hammer and Inverted Hammer candlestick patterns. The candles are constructed with five-minute SPY trade bars consolidated by a five-minute TradeBarConsolidator , while automatically updating with the RegisterIndicator method. Both patterns are bullish, so we buy SPY if the pattern is signaled.

C#

```
using QuantConnect.Indicators.CandlestickPatterns;
public class ConsolidatorUpdatingIndicatorsAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private Hammer _hammer = new();
    private InvertedHammer _invertedHammer = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request SPY data for signal generation and trading.
        _spy = AddEquity("SPY", Resolution.Minute).Symbol;

        // The candlestick patterns are based on a 5-minute consolidated trade bar.
        var consolidator = new TradeBarConsolidator(TimeSpan.FromMinutes(5));
        // Subscribe to update the indicators with the 5-minute consolidator automatically.
        RegisterIndicator(_spy, _hammer, consolidator);
        RegisterIndicator(_spy, _invertedHammer, consolidator);
        // Add event handler on candlestick indicators updated to trade the pattern.
        _hammer.Updated += OnUpdated;
        _invertedHammer.Updated += OnUpdated;

        SetWarmUp(1);
    }

    private void OnUpdated(object sender, IndicatorDataPoint point)
    {
        // Both the hammer and inverted Ham patterns indicate bullish movement, so we buy SPY.
        if (point.Value == 1 && !Portfolio[_spy].IsLong)
        {
            SetHoldings(_spy, 0.5m);
        }
    }

    public override void OnOrderEvent(OrderEvent orderEvent)
    {
        if (orderEvent.Status == OrderStatus.Filled)
        {
            if (orderEvent.Ticket.OrderType == OrderType.Market)
            {
                // Stop loss order at 1%.
                var stopPrice = orderEvent.FillQuantity > 0m ? orderEvent.FillPrice * 0.99m : orderEvent.FillPrice *
1.01m;
                StopMarketOrder(_spy, -Portfolio[_spy].Quantity, stopPrice);
                // Take profit order at 2%.
                var takeProfitPrice = orderEvent.FillQuantity > 0m ? orderEvent.FillPrice * 1.02m :
orderEvent.FillPrice * 0.98m;
                LimitOrder(_spy, -Portfolio[_spy].Quantity, takeProfitPrice);
            }
            else if (orderEvent.Ticket.OrderType == OrderType.StopMarket || orderEvent.Ticket.OrderType ==
OrderType.Limit)
            {
                // Cancel any open order if stop loss or take profit order filled.
                Transactions.CancelOpenOrders();
            }
        }
    }
}
```

Historical Data

Historical Data

Getting Started

Introduction

There are two ways to request historical data in your algorithms: direct historical data requests and indirect algorithm warm up .

History Requests

History requests return all of the data you request as a single object. Follow these steps to get some historical data:

1. Create a new project.

   Projects contain files to run backtests, launch research notebooks, perform parameter optimizations, and deploy live trading strategies. You need to create projects in order to create strategies and share your work with other members.

   The process to create a new project depends on if you use the Cloud Platform , Local Platform , or CLI .

2. In the `Initialize` method, add an asset .

   The `Initialize` method is the entry point of your algorithm where you define a series of settings, including security subscriptions, starting cash balances, and warm-up periods. LEAN only calls the `Initialize` method one time, at the start of your algorithm.

   ```
   C#
   var spy = AddEquity("SPY");
   ```

3. Call the `History` method with the asset's Symbol and a lookback period.

   `Symbol` objects are a way to identify or "finger-print" tradable assets so that no further database look-up is required. All QuantConnect and LEAN Algorithm API methods use `Symbol` objects to identify assets.

   ```
   C#
   var history = History(spy.Symbol, 5, Resolution.Daily);
   ```

Warm Up Periods

Warm-up simulates winding back the clock from the time you deploy the algorithm. In a backest, this is the start date of your algorithm. In live trading, it's the current date. Follow these steps to add a warm-up period to the start of your algorithm:

1. Create a new project.

   The process to create a new project depends on if you use the Cloud Platform , Local Platform , or CLI .

2. In the `Initialize` method, set the backtest dates and add an asset .

```
SetStartDate(2024, 12, 1);
SetEndDate(2024, 12, 2);
AddEquity("SPY", Resolution.Daily);
```

3. In the `Initialize` method, call the `SetWarmUp` method with the warm-up duration.

```
SetWarmUp(10, Resolution.Daily);
```

4. In the `OnData` method, log the time and warm-up state .

```
Log($"IsWarmingUp at {Time}: {IsWarmingUp}");
```

```
Algorithm starting warm up...
IsWarmingUp at 11/22/2024 4:00:00 PM: True
IsWarmingUp at 11/25/2024 4:00:00 PM: True
IsWarmingUp at 11/26/2024 4:00:00 PM: True
IsWarmingUp at 11/27/2024 4:00:00 PM: True
IsWarmingUp at 11/29/2024 1:00:00 PM: True
Algorithm finished warming up.
IsWarmingUp at 12/2/2024 4:00:00 PM: False
```

For more information about warm-up, see Warm Up Periods .

Examples

The following examples demonstrate some common practices for historical data.

Example 1: Simple Moving Average

The following example demonstrates weekly-resampled historical data to calculate the 50-week Simple Moving Average to implement an SMA-cross strategy.

C#

```
public class HistoricalDataAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private TradeBarConsolidator _consolidator;
    private RollingWindow<decimal> _window = new(50);

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Request SPY data for historical data call and trading.
        _spy = AddEquity("SPY").Symbol;
        // Set up a consolidator to resample historical data to weekly.
        _consolidator = new TradeBarConsolidator(Calendar.Weekly);
        _consolidator.DataConsolidated += (_, updated) => _window.Add(updated.Close);

        // Set a scheduled event to trade SPY with a weekly SMA signal.
        Schedule.On(DateRules.WeekStart(_spy), TimeRules.AfterMarketOpen(_spy, 1), Trade);
    }

    private void Trade()
    {
        // Historical data call to calculate signal.
        var history = History<TradeBar>(_spy, 252, Resolution.Daily);
        // Obtain the weekly close price to calculate the weekly EMA.
        foreach (var bar in history)
        {
            _consolidator.Update(bar);
        }
        // Calculate the current 50-week SMA as a trading signal.
        var sma50 = _window.Average();

        // Trade the SMA cross strategy.
        var price = history.Last().Close;
        if (price > sma50 && !Portfolio[_spy].IsLong)
        {
            SetHoldings(_spy, 1m);
        }
        else if (price < sma50 && !Portfolio[_spy].IsShort)
        {
            SetHoldings(_spy, -1m);
        }
    }
}
```

Historical Data

History Requests

Introduction

There are two ways to request historical data in your algorithms: direct historical data requests and indirect algorithm warm up . You can use a direct historical data request at any time throughout your algorithm. It returns all of the data you request as a single object.

Trailing Data Samples

To get historical data for a trailing time period, call `History` method with an an integer and a `Resolution` . For example, if you pass an asset `Symbol` , 5, and `Resolution.Minute` as the arguments, it returns the data of the asset during the most recent 5 minutes in the asset's market hours . These trailing minute bars can cross multiple trading days.

C#

```
public class TrailingDataSamplesHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of an asset.
        var symbol = AddEquity("SPY").Symbol;
        // Get the minute-resolution TradeBar data of the asset over the trailing 5 minutes.
        var history = History<TradeBar>(symbol, 5, Resolution.Minute);
    }
}
```

If you don't pass a `Resolution` , it defaults to the resolution of the security subscription. The following example returns 3 days of data for QQQ and 3 minutes of data for SPY:

C#

```
public class TrailingDataSamplesForMulitpleAssetsHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 20);
        SetEndDate(2024, 12, 31);
        // Add two assets with different resolutions.
        var spy = AddEquity("SPY", Resolution.Minute).Symbol;
        var qqq = AddEquity("QQQ", Resolution.Daily).Symbol;
        // Get the trailing 3 bars of data for each asset.
        var history = History<TradeBar>(new[] { spy, qqq }, 3);
    }
}
```

If there is no data for the time period you request, the result has fewer samples. For instance, say an illiquid asset has no trading activity during the last 15 minutes of the trading day and you request the 10 most recent minute bars at market close, 4 PM Eastern Standard Time (ET). In this case, you won't get any data because LEAN will try to fetch data from 3:50 PM ET to 4 PM ET since the market was open during that time , but there were no trades for the asset. For more information about missing data points, see Missing Data Points .

Trailing Time Periods

To get historical data for a trailing time period, pass a `TimeSpan` to the `History` method. The days of the `TimeSpan` represent

calendar days.

```
C#
public class TrailingTimePeriodHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of an asset.
        var symbol = AddEquity("SPY").Symbol;
        // Get the minute-resolution TradeBar data of the asset over the trailing 3 days.
        var history = History<TradeBar>(symbol, TimeSpan.FromDays(3), Resolution.Minute);
    }
}
```

If there is no data in the time period, the result is empty. For more information about missing data points, see Missing Data Points .

Date Ranges

To get historical data for a specific date range, call `History` method with start and end `DateTime` objects. The `DateTime` objects you provide are based in the algorithm time zone .

```
C#
public class DateRangeHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of an asset.
        var symbol = AddEquity("SPY").Symbol;
        // Get the daily-resolution TradeBar data of the asset during 2020.
        var history = History<TradeBar>(symbol, new DateTime(2020, 1, 1), new DateTime(2021, 1, 1),
Resolution.Daily);
    }
}
```

If there is no data for the date range you request, the result is empty. For more information about missing data points, see Missing Data Points .

Flat Universe DataFrames

To get a DataFrame of historical data, use Python.

Default Values

The following table describes the assumptions of the History API:

Argument	Assumption
`resolution`	LEAN guesses the resolution you request by looking at the securities you already have in your algorithm. If you have a security subscription in your algorithm with a matching `Symbol` , the history request uses the same resolution as the subscription. If you don't have a security subscription in your algorithm with a matching `Symbol` , `Resolution.Minute` is the default.
`<T>`	If you don't specify a type for the history request, `TradeBar` is the default. If the asset you request data for doesn't have `TradeBar` data, specify the `QuoteBar` type to receive history.

Additional Options

The `History` method accepts the following additional arguments:

Argument	Data Type	Description
`fillForward`	`bool?`	True to fill forward missing data. Otherwise, false.
`extendedMarketHours`	`bool?`	True to include extended market hours data. Otherwise, false.
`dataMappingMode`	`DataMappingMode?`	The contract mapping mode to use for the security history request.
`dataNormalizationMode`	`DataNormalizationMode?`	The price scaling mode to use for US Equities or continuous Futures contracts .
`contractDepthOffset`	`int?`	The desired offset from the current front month for continuous Futures contracts .

If you don't provide a value for some of the preceding options, it uses the option value from the security subscription.

```
C#
var future = AddFuture(Futures.Indices.SP500EMini);
var history = History(
    symbols: new[] {future.Symbol},
    start: Time - TimeSpan.FromDays(15),
    end: Time,
    resolution: Resolution.Minute,
    fillForward: false,
    extendedMarketHours: false,
    dataMappingMode: DataMappingMode.LastTradingDay,
    dataNormalizationMode: DataNormalizationMode.Raw,
    contractDepthOffset: 0);
```

Examples

The following examples demonstrate some common practices for trading using historical requests.

Example 1: Mean-Variance Portfolio

The following algorithm constructs a monthly rebalance mean-variance portfolio using the top 20 liquid equities. The position sizing can be optimized by 1-year historical daily return of the universe members.

C#

```
public class HistoricalRequestAlgorithm : QCAlgorithm
{
    private Universe _universe;

    // Instantiate the optimizer to perform mean-variance optimization.
    // Mean-variance optimization will not consider a risk-free rate, so we use 0.
    private readonly MaximumSharpeRatioPortfolioOptimizer _optimizer = new (0, 1, 0);

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Seed the last price to set the initial price.
        Settings.SeedInitialPrices = true;

        // Monthly renewal of the top 20 liquid universe to trade popular stocks.
        UniverseSettings.Schedule.On(DateRules.MonthStart());
        _universe = AddUniverse(Universe.Top(20));

        // Set a scheduled event to rebalance the portfolio at the start of every month.
        Schedule.On(DateRules.MonthStart(), TimeRules.At(9, 31), Rebalance);
    }

    private void Rebalance()
    {
        // Historical data request to get 1-year data for optimization.
        var symbols = _universe.Selected.ToList();
        var history = History<TradeBar>(symbols, 253, Resolution.Daily)
            .Where(x => symbols.All(y => x.ContainsKey(y)) && x.All(y => y.Value.Close > 0m))
            .ToList();
        // Daily return on the universe members to calculate the optimized weights.
        var returns = GetReturns(history, symbols);

        // Calculate the optimized weights.
        var weights = _optimizer.Optimize(returns);

        // Rebalance the portfolio according to the optimized weights (greater than 1%)
        var targets = Enumerable.Range(0, weights.Length)
            .Select(i =>
            {
                var weight = Math.Abs(weights[i]) > 0.01 ? Convert.ToDecimal(weights[i]) : 0m;
                return new PortfolioTarget(symbols[i], weight);
            })
            .ToList();
        SetHoldings(targets, liquidateExistingHoldings: true);
    }

    private static double[,] GetReturns(List<DataDictionary<TradeBar>> history, List<Symbol> symbols)
    {
        // Create a 2d array of historical daily returns from historical price data.
        var returns = new double[history.Count, symbols.Count];
        for (int j = 0; j < symbols.Count; j++)
        {
            var lastPrice = 0.0;
            for (int i = 0; i < history.Count; i++)
            {
                var current = (double) history[i][symbols[j]].Close;
                if (i > 0)
                {
                    returns[i, j] = (current / lastPrice) - 1;
                }
                lastPrice = current;
            }
        }
        return returns;
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm
HistoryAlgorithm.cs C#

Historical Data

History Responses

Introduction

When you make a history request, consider what data returns. Depending on how you request the data, history requests return a specific data type. The data that returns is in ascending order from oldest to newest. This order is necessary to use the data to warm up indicators.

DataFrames

To get a DataFrame from a history request, use Python.

Slices

To request `Slice` objects of historical data, call the `History` method. If you pass a list of `Symbol` objects, it returns data for all the securities and datasets that the `Symbol` objects reference.

```
C#
// Get the latest 3 data points of some securities/datasets, packaged into Slice objects.
var history = History(symbols, 3);
```

If you don't pass any `Symbol` objects, it returns data for all the data subscriptions in your notebook.

```
C#
// Get the latest 3 data points of all the securities/datasets in the notebook, packaged into Slice objects.
var history = History(3);
```

When your history request returns `Slice` objects, the `Time` properties of these objects are based on the algorithm time zone, but the `EndTime` properties of the individual data objects (for example, `TradeBar` , `QuoteBar` , and `Tick` objects) are based on the data time zone . The `EndTime` is the end of the sampling period and when the data is actually available. For more information about time periods, see Periods .

Lists

If you intend to use the data in the DataFrame to create objects like a `TradeBar` or `QuoteBar` , request that the history request returns the data type you need. Otherwise, LEAN consumes unnecessary computational resources populating the DataFrame. To request a list of objects containing the historical data, call one of the `History< Type >` methods.

```C#
public class ListHistoryResponseAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of an asset.
        var symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        // Create an indicator that is a function of historical bars.
        var indicator = new AverageTrueRange(21);
        // Get the latest TradeBar objects of the asset.
        var history = History<TradeBar>(symbol, indicator.WarmUpPeriod);
        // Update the indicator with the historical TradeBar objects.
        foreach (var bar in history)
        {
            indicator.Update(bar);
        }
    }
}
```

The methods return different data types, depending on how you call it. The following table shows the return type for each input:

Input Generic `< Type >`	**Input** `Symbol` **Type**	**Output Data Type**
`TradeBar`	`Symbol`	`List<TradeBar>`
`TradeBar`	`List<Symbol>`	`List<TradeBars>`
`QuoteBar`	`Symbol`	`List<QuoteBar>`
`QuoteBar`	`List<Symbol>`	`List<QuoteBars>`
`Tick`	`Symbol`	`List<Tick>`
`Tick`	`List<Symbol>`	`List<Ticks>`
`alternativeDataClass` (ex: `CBOE`)	`Symbol`	`List< alternativeDataClass >` (ex: `List<CBOE>`)
`alternativeDataClass` (ex: `CBOE`)	`List<Symbol>`	`List<Dictionary<Symbol, alternativeDataClass >>` (ex: `List<Dictionary<Symbol, CBOE>>`)

```C#
public class ListHistoryResponseAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of some assets.
        var spy = AddEquity("SPY").Symbol;
        var aapl = AddEquity("AAPL").Symbol;

        // Example: List<QuoteBars>
        foreach (var quoteBars in History<QuoteBar>(new[] { spy, aapl }, 3))
        {
            var t = quoteBars.Time;
            foreach (var kvp in quoteBars)
            {
                var symbol = kvp.Key;
                var quoteBar = kvp.Value;
                var bid = quoteBar.Bid;
            }
        }

        // Example: List<Ticks>
        foreach (var ticks in History(new[] { spy, aapl }, TimeSpan.FromDays(3), Resolution.Tick))
        {
            foreach (var kvp in ticks)
            {
                var symbol = kvp.Key;
                var tick = kvp.Value;
                var t = tick.Time;
                var price = tick.Price;
            }
        }

        // Example: List<Dictionary<Symbol, QuiverInsiderTrading>>
        var spyInsider = AddData<QuiverInsiderTrading>(spy).Symbol;
        var aaplInsider = AddData<QuiverInsiderTrading>(aapl).Symbol;
        foreach (var dataBySymbol in History<QuiverInsiderTrading>(new[] { spyInsider, aaplInsider }, 365,
Resolution.Daily))
        {
            foreach (var kvp in dataBySymbol)
            {
                var datasetSymbol = kvp.Key;
                var insiderTrading = kvp.Value;
                var assetSymbol = datasetSymbol.Underlying;
                var t = insiderTrading.EndTime;
                var sharesTraded = insiderTrading.Shares;
            }
        }
    }
}
```

When your history request returns a list of objects, the `Time` properties of these objects are based on the =$writingAlgorithms ? "algorithm" : "notebook" ?> time zone, but the `EndTime` properties of the individual objects in the list are based on the data time zone . The `EndTime` is the end of the sampling period and when the data is actually available. For more information about time periods, see Periods .

Missing Data Points

There are several cases where you may get fewer data points from a history request than you expect.

Illiquid Securities

The default behavior for history requests is to fill-forward the data. However, if an asset had no trading activity during the time period you request historical data for, you won't get any data for that asset.

Requesting Data Before an Asset's IPO

All assets have a day when they first started trading. If your history request includes time before an asset's initial public offering

(IPO) date, you won't get any data before the IPO date because the data doesn't exist.

Requesting Data Outside Market Hours

History requests for a trailing number of data samples return data based on the market hours of assets, so they can wrap across multiple trading days to gather the amount of data you request. In contrast, history requests for trailing time periods return data based on a rolling calendar time and history requests for specific date ranges return data for whatever range you request. In these two latter types of history requests, if the period of time is outside of the market hours of an asset, you won't get any data.

Data Issues

Data issues are generally a result of human error or from mistakes in the data collection process. Any QuantConnect member can report a data issue , which our Data Team works to quickly resolve. However, while the data is still missing, your history requests won't be able to return the data. To view the list of current data issues, log in to the Algorithm Lab and then, in the left navigation bar, click Datasets > Data Issues .

Historical Data

Asset Classes

Asset Classes

US Equities

Introduction

This page explains how to get historical data for US Equities. Some of the data you can get include prices, fundamentals, corporate actions, and universe selection data.

Trades

To get historical trade data , call the `History<TradeBar>` method with a security's `Symbol` .

```C#
public class USEquityTradeBarHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddEquity("SPY").Symbol;
        // Get the 5 trailing daily TradeBar objects of the security.
        var history = History<TradeBar>(symbol, 5, Resolution.Daily);
        // Iterate through each TradeBar and calculate its dollar volume.
        foreach (var bar in history)
        {
            var t = bar.EndTime;
            var dollarVolume = bar.Close * bar.Volume;
        }
    }
}
```

Quotes

To get historical quote data , call the `History<QuoteBar>` method with a security's `Symbol` .

```C#
public class USEquityQuoteBarHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddEquity("SPY").Symbol;
        // Get the 5 trailing minute QuoteBar objects of the security.
        var history = History<QuoteBar>(symbol, 5, Resolution.Minute);
        // Iterate through the QuoteBar objects and calculate the spread.
        foreach (var bar in history)
        {
            var t = bar.EndTime;
            var spread = bar.Ask.Close - bar.Bid.Close;
        }
    }
}
```

Request second or minute resolution data. Otherwise, the history request won't return any data.

Ticks

To get historical tick data , call the `History<Tick>` method with a security's `Symbol` and `Resolution.Tick` .

```C#
public class USEquityTickHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddEquity("SPY").Symbol;
        // Get the trailing 2 days of ticks for the security.
        var history = History<Tick>(symbol, TimeSpan.FromDays(2), Resolution.Tick);
        // Select the ticks that represent trades, excluding the quote ticks.
        var trades = history.Where(tick => tick.TickType == TickType.Trade);
    }
}
```

Ticks are a sparse dataset, so request ticks over a trailing period of time or between start and end times.

Slices

To get historical Slice data, call the `History` method without passing any `Symbol` objects. This method returns `Slice` objects, which contain data points from all the datasets in your algorithm. If you omit the `resolution` argument, it uses the resolution that you set for each security and dataset when you created the subscriptions.

```
C#
public class SliceHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);
        // Add some securities and datasets.
        AddEquity("SPY");
        // Get the historical Slice objects over the last 5 days for all the subcriptions in your algorithm.
        var history = History(5, Resolution.Daily);
        // Iterate through each historical Slice.
        foreach (var slice in history)
        {
            // Iterate through each TradeBar in this Slice.
            foreach (var kvp in slice.Bars)
            {
                var symbol = kvp.Key;
                var bar = kvp.Value;
            }
        }
    }
}
```

Fundamentals

To get historical fundamental data , call the `History<Fundamental>` method with an asset's `Symbol` .

```
C#
public class USEquityFundamentalHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 27);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of an asset.
        var symbol = AddEquity("AAPL").Symbol;
        // Get the 3 trailing daily Fundamental objects of the asset.
        var history = History<Fundamental>(symbol, 3, Resolution.Daily);
        // Iterate through each Fundamental object.
        foreach (var fundamental in history)
        {
            var t = fundamental.EndTime;
            var currentRatio = fundamental.OperationRatios.CurrentRatio.Value;
        }
    }
}
```

Splits

To get historical split data , call the `History<Split>` method with an asset's `Symbol` . Splits are a sparse dataset, so use a time period history request since most days have no data.

```
C#
public class USEquitySplitHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of an asset.
        var symbol = AddEquity("AAPL").Symbol;
        // Get the splits that occured for the stock over the last 5 years.
        var history = History<Split>(symbol, TimeSpan.FromDays(5*365));
        // Select the dates when splits occurred.
        var splitDates = history.Where(split => split.Type == SplitType.SplitOccurred).Select(split =>
split.EndTime);
    }
}
```

Dividends

To get historical dividend data , call the `History<Dividend>` method with an asset's `Symbol` . Dividends are a sparse dataset, so use a time period history request since most days have no data.

```
C#
public class USEquityDividendHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of an asset.
        var symbol = AddEquity("AAPL").Symbol;
        // Get the dividends that the stock paid over the last 2 years.
        var history = History<Dividend>(symbol, TimeSpan.FromDays(2*365));
        // Calculate the mean dividend payment.
        var meanDividend = history.Average(split => split.Distribution);
    }
}
```

Symbol Changes

To get historical symbol change data , call the `History<SymbolChangedEvent>` method with an asset's `Symbol` .

```
C#
public class USEquitySymbolChangedEventHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of an asset.
        var symbol = AddEquity("META").Symbol;
        // Get the symbol changes of the stock over the last 10 years.
        var history = History<SymbolChangedEvent>(symbol, TimeSpan.FromDays(10*365));
        // Iterate through each SymbolChangedEvent object to access their data point attributes.
        foreach (var symbolChangedEvent in history)
        {
            var t = symbolChangedEvent.EndTime;
            var oldSymbol = symbolChangedEvent.OldSymbol;
            var newSymbol = symbolChangedEvent.NewSymbol;
        }
    }
}
```

Delistings

To get historical delisting data , call the `History<Delisting>` method with an asset's `Symbol` . Delistings are a sparse dataset, so use a time period history request since most days have no data.

```
C#
public class USEquityDelistingHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of an asset.
        var symbol = AddEquity("BBBY").Symbol;
        // Get the deslistings of the asset over the last 10 years.
        var history = History<Delisting>(symbol, TimeSpan.FromDays(10*365));
        // Get the dates of the delist warnings.
        var delistWarningDates = history
            .Where(delisting => delisting.Type == DelistingType.Warning)
            .Select(delisting => delisting.EndTime);
    }
}
```

Universes

To get historical universe data , call the `History` method with the `Universe` object.

```
C#
public class USEquityUniverseHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 23);
        SetEndDate(2024, 12, 31);
        // Add a universe of US Equities based on the constituents of an ETF.
        var universe = AddUniverse(Universe.ETF("SPY"));
        // Get 5 days of history for the universe.
        var history = History(universe, TimeSpan.FromDays(5));
        // Iterate through each day of the universe history.
        foreach (var constituents in history)
        {
            // Select the 2 assets with the smallest weights in the ETF on this day.
            var dailyLargest = constituents.Select(c => c as ETFConstituentData).OrderByDescending(c =>
c.Weight).Take(2);
        }
    }
}
```

Alternative Data

To get historical alternative data, call the `History< alternativeDataClass >` method with the dataset `Symbol` .

```
C#
public class USEquityAlternativeDataHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of an asset.
        var symbol = AddEquity("GME").Symbol;
        // Add the alternative dataset and save a reference to its Symbol.
        var datasetSymbol = AddData<QuiverWallStreetBets>(symbol).Symbol;
        // Get the trailing 5 days of QuiverWallStreetBets data for the asset.
        var history = History<QuiverWallStreetBets>(datasetSymbol, 5, Resolution.Daily);
        // Iterate each data point and access its attributes.
        foreach (var dataPoint in history)
        {
            var t = dataPoint.EndTime;
            var sentiment = dataPoint.Sentiment;
        }
    }
}
```

Some alternative datasets provide multiple entries per asset per time step . For example, the True Beats dataset provides EPS and revenue predictions for several upcoming quarters per asset per day. In this case, use a nested loop to access all the data point attributes.

```
C#
public class USEquityTrueBeatsHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Get the ExtractAlphaTrueBeats data for AAPL on 01/02/2024.
        var symbol = AddEquity("AAPL", Resolution.Daily).Symbol;
        var datasetSymbol = AddData<ExtractAlphaTrueBeats>(symbol).Symbol;
        var history = History<ExtractAlphaTrueBeats>(
            datasetSymbol, new DateTime(2024, 9, 1), new DateTime(2024, 12, 31), Resolution.Daily
        );
        // Iterate through each day of history.
        foreach (var trueBeats in history)
        {
            // Calculate the mean TrueBeat estimate for this day.
            var t = trueBeats.EndTime;
            var meanTrueBeat = trueBeats.Data
                .Select(estimate => estimate as ExtractAlphaTrueBeat)
                .Average(estimate => estimate.TrueBeat);
        }
    }
}
```

For information on historical data for other alternative datasets, see the documentation in the Dataset Market .

Indicators

To get historical indicator values, call the `IndicatorHistory` method with an indicator and the security's `Symbol` .

```
C#
public class USEquityIndicatorHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddEquity("SPY").Symbol;
        // Get the 21-day SMA values of the security for the last 5 trading days.
        var history = IndicatorHistory(new SimpleMovingAverage(21), symbol, 5, Resolution.Daily);
        // Get the maximum of the SMA values.
        var maxSMA = history.Max(indicatorDataPoint => indicatorDataPoint.Current.Value);
    }
}
```

The `IndicatorHistory` method resets your indicator, makes a history request, and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

To make the `IndicatorHistory` method update the indicator with an alternative price field instead of the close (or mid-price) of each bar, pass a `selector` argument.

```
C#
// Get the historical values of an indicator over the last 30 days, applying the indicator to the security's volume.
var history = IndicatorHistory(indicator, symbol, TimeSpan.FromDays(30), selector: Field.Volume);
```

Some indicators require the prices of two securities to compute their value (for example, Beta). In this case, pass a list of the `Symbol` objects to the method.

```
C#
public class USEquityMultiAssetIndicatorHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Add the target and reference securities.
        var targetSymbol = AddEquity("AAPL").Symbol;
        var referenceSymbol = AddEquity("SPY").Symbol;
        // Create a 21-period Beta indicator.
        var beta = new Beta("", targetSymbol, referenceSymbol, 21);
        // Get the historical values of the indicator over the last 10 trading days.
        var history = IndicatorHistory(beta, new[] {targetSymbol, referenceSymbol}, 10, Resolution.Daily);
        // Get the average Beta value.
        var avgBeta = history.Average(indicatorDataPoint => indicatorDataPoint.Current.Value);
    }
}
```

If you already have a list of Slice objects, you can pass them to the `IndicatorHistory` method to avoid the internal history request.

```
C#
var slices = History(new[] {symbol}, 30, Resolution.Daily);
var history = IndicatorHistory(indicator, slices);
```

Short Availability

To get historical values for the borrow fee rate, borrow rebate rate, or shortable quantity of a US Equity, call the `FeeRate` , `RebateRate` , or `ShortableQuantity` methods of its shortable provider .

```
C#
public class USEquityShortAvailabilityHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Add an asset and save a reference to the Equity object.
        var security = AddEquity("SPY");
        // Overwrite the default shortable provider to one that provides data.
        security.SetShortableProvider(new InteractiveBrokersShortableProvider());
        // Select a time in the past.
        var t = Time.AddDays(-30);
        // Pass the time argument to shortable provider to get historical values.
        var feeRate = security.ShortableProvider.FeeRate(security.Symbol, t);
        var rebateRate = security.ShortableProvider.RebateRate(security.Symbol, t);
        var shortableQuantity = security.ShortableProvider.ShortableQuantity(security.Symbol, t);
    }
}
```

For more information about security initializers, see Set Security Initializer .

Examples

The following examples demonstrate some common practices for trading using historical requests.

Example 1: Mean-Variance Portfolio

The following algorithm constructs a monthly rebalance mean-variance portfolio using the top 20 liquid equities. The position sizing can be optimized by 1-year historical daily return of the universe members.

C#

```
public class HistoricalRequestAlgorithm : QCAlgorithm
{
    private Universe _universe;

    // Instantiate the optimizer to perform mean-variance optimization.
    // Mean-variance optimization will not consider a risk-free rate, so we use 0.
    private readonly MaximumSharpeRatioPortfolioOptimizer _optimizer = new (0, 1, 0);

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Seed the last price to set the initial price.
        Settings.SeedInitialPrices = true;

        // Monthly renewal of the top 20 liquid universe to trade popular stocks.
        UniverseSettings.Schedule.On(DateRules.MonthStart());
        _universe = AddUniverse(Universe.Top(20));

        // Set a scheduled event to rebalance the portfolio at the start of every month.
        Schedule.On(DateRules.MonthStart(), TimeRules.At(9, 31), Rebalance);
    }

    private void Rebalance()
    {
        // Historical data request to get 1-year data for optimization.
        var symbols = _universe.Selected.ToList();
        var history = History<TradeBar>(symbols, 253, Resolution.Daily)
            .Where(x => symbols.All(y => x.ContainsKey(y)) && x.All(y => y.Value.Close > 0m))
            .ToList();
        // Daily return on the universe members to calculate the optimized weights.
        var returns = GetReturns(history, symbols);

        // Calculate the optimized weights.
        var weights = _optimizer.Optimize(returns);

        // Rebalance the portfolio according to the optimized weights (greater than 1%)
        var targets = Enumerable.Range(0, weights.Length)
            .Select(i =>
            {
                var weight = Math.Abs(weights[i]) > 0.01 ? Convert.ToDecimal(weights[i]) : 0m;
                return new PortfolioTarget(symbols[i], weight);
            })
            .ToList();
        SetHoldings(targets, liquidateExistingHoldings: true);
    }

    private static double[,] GetReturns(List<DataDictionary<TradeBar>> history, List<Symbol> symbols)
    {
        // Create a 2d array of historical daily returns from historical price data.
        var returns = new double[history.Count, symbols.Count];
        for (int j = 0; j < symbols.Count; j++)
        {
            var lastPrice = 0.0;
            for (int i = 0; i < history.Count; i++)
            {
                var current = (double) history[i][symbols[j]].Close;
                if (i > 0)
                {
                    returns[i, j] = (current / lastPrice) - 1;
                }
                lastPrice = current;
            }
        }
        return returns;
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm
HistoryAlgorithm.cs C#

Asset Classes

Equity Options

Introduction

This page explains how to get historical data for Equity Options. Some of the data you can get include prices, open interest, Greeks, and universe selection data.

Trades

To get historical trade data , call the `History<TradeBar>` method with a security's `Symbol` .

C#

```
public class EquityOptionTradeBarHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var equity = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw);
        var symbol = OptionChain(equity.Symbol).OrderBy(c => c.OpenInterest).Last().Symbol;
        // Get the 5 trailing daily TradeBar objects of the security.
        var history = History<TradeBar>(symbol, 5, Resolution.Daily);
        // Iterate through each TradeBar and calculate its dollar volume.
        foreach (var bar in history)
        {
            var t = bar.EndTime;
            var dollarVolume = bar.Close * bar.Volume;
        }
    }
}
```

Request minute, hour, or daily resolution data. Otherwise, the history request won't return any data.

Quotes

To get historical quote data , call the `History<QuoteBar>` method with a security's `Symbol` .

C#

```
public class EquityOptionsQuoteBarHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var equity = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw);
        var symbol = OptionChain(equity.Symbol).OrderBy(c => c.OpenInterest).Last().Symbol;
        // Get the 5 trailing minute QuoteBar objects of the security.
        var history = History<QuoteBar>(symbol, 5, Resolution.Minute);
        // Iterate through the QuoteBar objects and calculate the spread.
        foreach (var bar in history)
        {
            var t = bar.EndTime;
            var spread = bar.Ask.Close - bar.Bid.Close;
        }
    }
}
```

Request minute, hour, or daily resolution data. Otherwise, the history request won't return any data.

Slices

To get historical Slice data, call the `History` method without passing any `Symbol` objects. This method returns `Slice` objects, which contain data points from all the datasets in your algorithm. If you omit the `resolution` argument, it uses the resolution that you set for each security and dataset when you created the subscriptions.

```
C#
public class SliceHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);
        // Add some securities and datasets.
        var equity = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw);
        var contract = OptionChain(equity.Symbol).OrderBy(c => c.OpenInterest).Last();
        AddOptionContract(contract.Symbol);
        // Get the historical Slice objects over the last 5 days for all the subcriptions in your algorithm.
        var history = History(5, Resolution.Daily);
        // Iterate through each historical Slice.
        foreach (var slice in history)
        {
            // Iterate through each TradeBar in this Slice.
            foreach (var kvp in slice.Bars)
            {
                var symbol = kvp.Key;
                var bar = kvp.Value;
            }
        }
    }
}
```

Open Interest

To get historical open interest data , call the `History<OpenInterest>` method with a security's `Symbol` .

```
C#
public class EquityOptionOpenInterestHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var equity = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw);
        var symbol = OptionChain(equity.Symbol).OrderBy(c => c.OpenInterest).Last().Symbol;
        // Get the 5 trailing daily OpenInterest objects of the security.
        var history = History<OpenInterest>(symbol, 5, Resolution.Daily);
        // Iterate through each OpenInterest and get it's value.
        foreach (var oi in history)
        {
            var t = oi.EndTime;
            var openInterest = oi.Value;
        }
    }
}
```

Daily Option Chains

To get historical daily Option chain data , call the `History` method with the Option `Symbol` object. The data this method returns contains information on all the currently tradable contracts, not just the contracts that pass your filter.

```C#
public class EquityOptionDailyOptionChainHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 23);
        SetEndDate(2024, 12, 31);
        // Add an Equity Option universe.
        var option = AddOption("SPY");
        // Get the trailing 5 daily Option chains.
        var history = History<OptionUniverse>(option.Symbol, 5);
        // Iterate through each day of the history.
        foreach (var optionUniverse in history)
        {
            var t = optionUniverse.EndTime;
            // Select the 2 contracts with the most volume.
            var mostTraded = optionUniverse.Select(c => c as OptionUniverse).OrderByDescending(c =>
c.Volume).Take(2);
        }
    }
}
```

Indicators

To get historical indicator values, call the `IndicatorHistory` method with an indicator and the security's `Symbol` .

```C#
public class EquityOptionIndicatorHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of the Option contract.
        var underlying = AddEquity("SPY").Symbol;
        var symbol = OptionChain(underlying).OrderBy(c => c.OpenInterest).Last().Symbol;
        // Get the 21-day SMA values of the contract for the last 5 trading days.
        var history = IndicatorHistory(new SimpleMovingAverage(21), symbol, 5, Resolution.Daily);
        // Get the maximum of the SMA values.
        var maxSMA = history.Max(indicatorDataPoint => indicatorDataPoint.Current.Value);
    }
}
```

The `IndicatorHistory` method resets your indicator, makes a history request, and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

To make the `IndicatorHistory` method update the indicator with an alternative price field instead of the close (or mid-price) of each bar, pass a `selector` argument.

```C#
// Get the historical values of an indicator over the last 30 days, applying the indicator to the security's volume.
var history = IndicatorHistory(indicator, symbol, TimeSpan.FromDays(30), selector: Field.Volume);
```

Some indicators require the prices of multiple securities to compute their value (for example, the indicators for the Greeks and implied volatility). In this case, pass a list of the `Symbol` objects to the method.

```C#
public class EquityOptionMultiAssetIndicatorHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of the underlying asset.
        var underlying = AddEquity("SPY").Symbol;
        // Get the Option contract Symbol.
        var option = OptionChain(underlying).OrderBy(c => c.OpenInterest).Last().Symbol;
        // Get the Symbol of the mirror contract.
        var mirror = QuantConnect.Symbol.CreateOption(
            underlying, option.ID.Market, option.ID.OptionStyle,
            option.ID.OptionRight == OptionRight.Put ? OptionRight.Call : OptionRight.Put,
            option.ID.StrikePrice, option.ID.Date
        );
        // Create the indicator.
        var indicator = new ImpliedVolatility(option, RiskFreeInterestRateModel, new
DividendYieldProvider(underlying), mirror, OptionPricingModelType.ForwardTree);
        // Get the historical values of the indicator over the last 60 trading minutes.
        var history = IndicatorHistory(indicator, new[] {underlying, option, mirror}, 60, Resolution.Minute);
        // Get the average IV value.
        var avgIV = history.Average(indicatorDataPoint => indicatorDataPoint.Current.Value);
    }
}
```

If you already have a list of Slice objects, you can pass them to the `IndicatorHistory` method to avoid the internal history request.

```C#
var slices = History(new[] {symbol}, 30, Resolution.Daily);
var history = IndicatorHistory(indicator, slices);
```

Examples

The following examples demonstrate some common practices for trading using historical requests.

Example 1: Trend Following on 0DTE Option Contract

This algorithm strategically trades 0DTE SPY options by analyzing bid and ask volumes shortly after the market opens. Using scheduled events, it effectively executes trades based on historical quote data, optimizing decision-making. The algorithm aims for timely entries and exits, ensuring efficient capital management and quick adaptability in volatile markets.

C#

```
public class ZeroDTEOptionsTradingAlgorithm : QCAlgorithm
{
    private Symbol _option;

    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 10);
        SetCash(100000);

        // Request SPY option data for trading and signal generation.
        var option = AddOption("SPY");
        // We are interested in ATM 0DTE options since they are the most popular.
        option.SetFilter(u => u.IncludeWeeklys().Expiration(0, 0).Strikes(-3, 3));
        _option = option.Symbol;

        // Schedule event to enter and exit option contract position.
        Schedule.On(DateRules.EveryDay(_option), TimeRules.AfterMarketOpen(_option, 16), TradeOption);
        Schedule.On(DateRules.EveryDay(_option), TimeRules.BeforeMarketClose(_option, 15), () => Liquidate());
    }

    public void TradeOption()
    {
        // Get the option chain for SPY to trade.
        if (CurrentSlice.OptionChains.TryGetValue(_option, out var optionChain))
        {
            foreach (var option in optionChain)
            {
                // Request historical quote data for signal generation.
                var history = History<QuoteBar>(option.Symbol, 15, Resolution.Minute);
                // Calculate the total bid and ask for dollar volume to determine the capital directional force.
                var totalBidVolume = 0m;
                var totalAskVolume = 0m;
                foreach (var bar in history)
                {
                    totalBidVolume += bar.Bid.Close * bar.LastBidSize;
                    totalAskVolume += bar.Ask.Close * bar.LastAskSize;
                }

                // Follow the capital flow to trade.
                if (totalBidVolume > totalAskVolume)
                {
                    MarketOrder(option.Symbol, 1);
                }
                else
                {
                    MarketOrder(option.Symbol, -1);
                }
            }
        }
    }
}
```

Asset Classes

Crypto

Introduction

This page explains how to get historical data for Crypto. Some of the data you can get include prices, indicators, and universe selection data.

Trades

To get historical trade data , call the `History<TradeBar>` method with a security's `Symbol` .

```
C#
public class CryptoTradeBarHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddCrypto("BTCUSD").Symbol;
        // Get the 5 trailing daily TradeBar objects of the security.
        var history = History<TradeBar>(symbol, 5, Resolution.Daily);
        // Iterate through each TradeBar and calculate its dollar volume.
        foreach (var bar in history)
        {
            var t = bar.EndTime;
            var dollarVolume = bar.Close * bar.Volume;
        }
    }
}
```

Quotes

To get historical quote data , call the `History<QuoteBar>` method with a security's `Symbol` .

```
C#
public class CryptoQuoteBarHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddCrypto("BTCUSD", market: Market.Bitfinex).Symbol;
        // Get the 5 trailing minute QuoteBar objects of the security.
        var history = History<QuoteBar>(symbol, 5, Resolution.Minute);
        // Iterate through the QuoteBar objects and calculate the spread.
        foreach (var bar in history)
        {
            var t = bar.EndTime;
            var spread = bar.Ask.Close - bar.Bid.Close;
        }
    }
}
```

Ticks

To get historical tick data , call the `History<Tick>` method with a security's `Symbol` and `Resolution.Tick` .

```C#
public class CryptoTickHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddCrypto("BTCUSD", market: Market.Bitfinex).Symbol;
        // Get the trailing 2 days of ticks for the security.
        var history = History<Tick>(symbol, TimeSpan.FromDays(2), Resolution.Tick);
        // Select the ticks that represent trades, excluding the quote ticks.
        var trades = history.Where(tick => tick.TickType == TickType.Trade);
    }
}
```

Ticks are a sparse dataset, so request ticks over a trailing period of time or between start and end times.

Slices

To get historical Slice data, call the `History` method without passing any `Symbol` objects. This method returns `Slice` objects, which contain data points from all the datasets in your algorithm. If you omit the `resolution` argument, it uses the resolution that you set for each security and dataset when you created the subscriptions.

```C#
public class SliceHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);
        // Add some securities and datasets.
        AddCrypto("BTCUSD");
        // Get the historical Slice objects over the last 5 days for all the subcriptions in your algorithm.
        var history = History(5, Resolution.Daily);
        // Iterate through each historical Slice.
        foreach (var slice in history)
        {
            // Iterate through each TradeBar in this Slice.
            foreach (var kvp in slice.Bars)
            {
                var symbol = kvp.Key;
                var bar = kvp.Value;
            }
        }
    }
}
```

Universes

To get historical universe data , call the `History` method with the `Universe` object. This method doesn't apply your selection function . It returns all of the universe data.

```
C#
public class CryptoUniverseHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 23);
        SetEndDate(2024, 12, 31);
        // Add a universe of Cryptocurrencies on Coinbase.
        var universe = AddUniverse(CryptoUniverse.Coinbase(constituents => constituents.Select(c => c.Symbol)));
        // Get 5 days of history for the universe.
        var history = History(universe, TimeSpan.FromDays(5));
        // Iterate through each day of the universe history.
        foreach (var constituents in history)
        {
            // Select the 2 assets with the largest dollar volume on this day.
            var dailyLargest = constituents.Select(c => c as CryptoUniverse).OrderByDescending(c =>
c.VolumeInUsd).Take(2);
        }
    }
}
```

Alternative Data

To get historical alternative data, call the `History< alternativeDataClass >` method with the dataset `Symbol` .

```
C#
public class CryptoAlternativeDataHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of an asset.
        var symbol = AddEquity("BTCUSD").Symbol;
        // Add the alternative dataset and save a reference to its Symbol.
        var datasetSymbol = AddData<BitcoinMetadata>(symbol).Symbol;
        // Get the trailing 5 days of alternative data for the asset.
        var history = History<BitcoinMetadata>(datasetSymbol, 5, Resolution.Daily);
        // Iterate each data point and access its attributes.
        foreach (var dataPoint in history)
        {
            var t = dataPoint.EndTime;
            var marketCap = dataPoint.MarketCapitalization;
        }
    }
}
```

For information on historical data for other alternative datasets, see the documentation in the Dataset Market .

Indicators

To get historical indicator values, call the `IndicatorHistory` method with an indicator and the security's `Symbol` .

```
C#
public class CryptoIndicatorHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddCrypto("BTCUSD").Symbol;
        // Get the 21-day SMA values of the security for the last 5 trading days.
        var history = IndicatorHistory(new SimpleMovingAverage(21), symbol, 5, Resolution.Daily);
        // Get the maximum of the SMA values.
        var maxSMA = history.Max(indicatorDataPoint => indicatorDataPoint.Current.Value);
    }
}
```

The `IndicatorHistory` method resets your indicator, makes a history request, and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

To make the `IndicatorHistory` method update the indicator with an alternative price field instead of the close (or mid-price) of each bar, pass a `selector` argument.

```
C#
// Get the historical values of an indicator over the last 30 days, applying the indicator to the security's volume.
var history = IndicatorHistory(indicator, symbol, TimeSpan.FromDays(30), selector: Field.Volume);
```

Some indicators require the prices of two securities to compute their value (for example, Beta). In this case, pass a list of the `Symbol` objects to the method.

```
C#
public class CryptoMultiAssetIndicatorHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Add the target and reference securities.
        var targetSymbol = AddCrypto("DOGEUSD").Symbol;
        var referenceSymbol = AddCrypto("BTCUSD").Symbol;
        // Create a 21-period Beta indicator.
        var beta = new Beta("", targetSymbol, referenceSymbol, 21);
        // Get the historical values of the indicator over the last 10 trading days.
        var history = IndicatorHistory(beta, new[] {targetSymbol, referenceSymbol}, 10, Resolution.Daily);
        // Get the average Beta value.
        var avgBeta = history.Average(indicatorDataPoint => indicatorDataPoint.Current.Value);
    }
}
```

If you already have a list of Slice objects, you can pass them to the `IndicatorHistory` method to avoid the internal history request.

```
C#
var slices = History(new[] {symbol}, 30, Resolution.Daily);
var history = IndicatorHistory(indicator, slices);
```

Examples

The following examples demonstrate some common practices for trading Cryptos using historical requests.

Example 1: Risk-Parity Crypto Pairs Portfolio

The following algorithm constructs a monthly rebalance risk-parity portfolio using the top 10 liquid Crypto pairs in the BitFinex market. A risk Parity portfolio can dissipate the risk of each crypto pair to reduce overall drawdown. The position sizing can be optimized by 1-year historical daily return of the universe members.

C#

```
public class CryptoHistoricalDataAlgorithm : QCAlgorithm
{
    private Universe _universe;
    // Instantiate the optimizer to perform risk-parity optimization.
    // Risk Parity portfolio can dissipate the risk of each crypto pair to reduce overall drawdown.
    private readonly RiskParityPortfolioOptimizer _optimizer = new(0, 1);

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Seed the last price to set the initial price of the BTCUSDT holdings.
        Settings.SeedInitialPrices = true;

        // Monthly renewal of the top 10 liquid universe to trade popular crypto pairs.
        UniverseSettings.Schedule.On(DateRules.MonthStart());
        _universe = AddUniverse(new CryptoCoarseFundamentalUniverse(Market.Bitfinex, UniverseSettings, Selection));

        // Set a scheduled event to rebalance the portfolio at the start of every month.
        Schedule.On(DateRules.MonthStart(), TimeRules.At(9, 31), Rebalance);
    }

    private IEnumerable<Symbol> Selection(IEnumerable<CryptoUniverse> data)
    {
        // Filter out the crypto pair with a price below $10 to avoid excessive volatility.
        return (from datum in data
                where datum.VolumeInUsd.HasValue
                orderby datum.VolumeInUsd.Value descending
                select datum.Symbol).Take(10);
    }

    private void Rebalance()
    {
        // Historical data request to get 1-year data for optimization.
        var symbols = _universe.Selected.ToList();
        var history = History<TradeBar>(symbols, 253, Resolution.Daily)
            .Where(x => symbols.All(y => x.ContainsKey(y)) && x.All(y => y.Value.Close > 0m))
            .ToList();
        // Daily return on the universe members to calculate the optimized weights.
        var returns = GetReturns(history, symbols);

        // Calculate the optimized weights.
        var weights = _optimizer.Optimize(returns);

        // Rebalance the portfolio according to the optimized weights.
        var targets = Enumerable.Range(0, weights.Length)
            .Select(i =>
            {
                var weight = Math.Abs(weights[i]) > 0.01 ? Convert.ToDecimal(weights[i]) : 0m;
                return new PortfolioTarget(symbols[i], weight);
            })
            .ToList();
        SetHoldings(targets, liquidateExistingHoldings: true);
    }

    private static double[,] GetReturns(List<DataDictionary<TradeBar>> history, List<Symbol> symbols)
    {
        // Create a 2d array of historical daily returns from historical price data.
        var returns = new double[history.Count, symbols.Count];
        for (int j = 0; j < symbols.Count; j++)
        {
            var lastPrice = 0.0;
            for (int i = 0; i < history.Count; i++)
            {
                var current = (double) history[i][symbols[j]].Close;
                if (i > 0)
                {
                    returns[i, j] = current / lastPrice - 1;
                }
                lastPrice = current;
            }
        }
        return returns;
    }
}
```

Asset Classes

Crypto Futures

Introduction

This page explains how to get historical data for Crypto Futures. Some of the data you can get include prices, margin interest rates, and indicator data.

Trades

To get historical trade data , call the `History<TradeBar>` method with a security's `Symbol` .

```
C#
public class CryptoFutureTradeBarHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddCryptoFuture("BTCUSD").Symbol;
        // Get the 5 trailing daily TradeBar objects of the security.
        var history = History<TradeBar>(symbol, 5, Resolution.Daily);
        // Iterate through each TradeBar and calculate its dollar volume.
        foreach (var bar in history)
        {
            var t = bar.EndTime;
            var dollarVolume = bar.Close * bar.Volume;
        }
    }
}
```

Quotes

To get historical quote data , call the `History<QuoteBar>` method with a security's `Symbol` .

```
C#
public class CryptoFutureQuoteBarHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddCryptoFuture("BTCUSD").Symbol;
        // Get the 5 trailing minute QuoteBar objects of the security.
        var history = History<QuoteBar>(symbol, 5, Resolution.Minute);
        // Iterate through the QuoteBar objects and calculate the spread.
        foreach (var bar in history)
        {
            var t = bar.EndTime;
            var spread = bar.Ask.Close - bar.Bid.Close;
        }
    }
}
```

Ticks

To get historical tick data , call the `History<Tick>` method with a security's `Symbol` and `Resolution.Tick` .

```
C#
public class CryptoFutureTickHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddCryptoFuture("BTCUSD").Symbol;
        // Get the trailing 2 days of ticks for the security.
        var history = History<Tick>(symbol, TimeSpan.FromDays(2), Resolution.Tick);
        // Select the ticks that represent trades, excluding the quote ticks.
        var trades = history.Where(tick => tick.TickType == TickType.Trade);
    }
}
```

Ticks are a sparse dataset, so request ticks over a trailing period of time or between start and end times.

Slices

To get historical Slice data, call the `History` method without passing any `Symbol` objects. This method returns `Slice` objects, which contain data points from all the datasets in your algorithm. If you omit the `resolution` argument, it uses the resolution that you set for each security and dataset when you created the subscriptions.

```
C#
public class SliceHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);
        // Add some securities and datasets.
        AddCryptoFuture("BTCUSD");
        // Get the historical Slice objects over the last 5 days for all the subcriptions in your algorithm.
        var history = History(5, Resolution.Daily);
        // Iterate through each historical Slice.
        foreach (var slice in history)
        {
            // Iterate through each TradeBar in this Slice.
            foreach (var kvp in slice.Bars)
            {
                var symbol = kvp.Key;
                var bar = kvp.Value;
            }
        }
    }
}
```

Margin Interest Rates

To get historical margin interest rate data , call the `History<MarginInterestRate>` method with a security's `Symbol` .

```C#
public class CryptoFutureMarginInterestRateHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddCryptoFuture("BTCUSD").Symbol;
        // Get the MarginInterestRate objects of the security over the last 2 trading days.
        var history = History<MarginInterestRate>(symbol, 2, Resolution.Daily);
        // Iterate through each MarginInterestRate object and get its value.
        foreach (var dataPoint in history)
        {
            var t = dataPoint.EndTime;
            var interestRate = dataPoint.InterestRate;
        }
    }
}
```

Indicators

To get historical indicator values, call the `IndicatorHistory` method with an indicator and the security's `Symbol` .

```C#
public class CryptoFutureIndicatorHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddCryptoFuture("BTCUSD").Symbol;
        // Get the 21-day SMA values of the security for the last 5 trading days.
        var history = IndicatorHistory(new SimpleMovingAverage(21), symbol, 5, Resolution.Daily);
        // Get the maximum of the SMA values.
        var maxSMA = history.Max(indicatorDataPoint => indicatorDataPoint.Current.Value);
    }
}
```

The `IndicatorHistory` method resets your indicator, makes a history request, and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

To make the `IndicatorHistory` method update the indicator with an alternative price field instead of the close (or mid-price) of each bar, pass a `selector` argument.

```C#
// Get the historical values of an indicator over the last 30 days, applying the indicator to the security's volume.
var history = IndicatorHistory(indicator, symbol, TimeSpan.FromDays(30), selector: Field.Volume);
```

Some indicators require the prices of two securities to compute their value (for example, Beta). In this case, pass a list of the `Symbol` objects to the method.

```C#
public class CryptoFutureMultiAssetIndicatorHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Add the target and reference securities.
        var targetSymbol = AddCryptoFuture("DOGEUSD").Symbol;
        var referenceSymbol = AddCryptoFuture("BTCUSD").Symbol;
        // Create a 21-period Beta indicator.
        var beta = new Beta("", targetSymbol, referenceSymbol, 21);
        // Get the historical values of the indicator over the last 10 trading days.
        var history = IndicatorHistory(beta, new[] {targetSymbol, referenceSymbol}, 10, Resolution.Daily);
        // Get the average Beta value.
        var avgBeta = history.Average(indicatorDataPoint => indicatorDataPoint.Current.Value);
    }
}
```

If you already have a list of Slice objects, you can pass them to the `IndicatorHistory` method to avoid the internal history request.

```C#
var slices = History(new[] {symbol}, 30, Resolution.Daily);
var history = IndicatorHistory(indicator, slices);
```

Examples

The following examples demonstrate common practices for trading cryptocurrency futures with historical data.

Example 1: Spot-Future Arbitrage

The example below uses linear regression with historical Spot and Future BTCUSDT data to obtain the cointegrating vector between them. We measure the spread between the two securities and trade mean reversal upon their spread divergence.

```C#
using MathNet.Numerics.LinearRegression;
using Accord.Math;

public class CointegrationVectorRegressionAlgorithm : QCAlgorithm
{
    private Symbol _spot, _future;
    private decimal[] _cointegrationVector = new[] { 0m, 0m };
    private decimal _residualMean = 0m;
    private decimal _residualStd = 1m;
    private decimal _entryThreshold = 2m;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetAccountCurrency("USDT", 100000);
        SetCash("BTC", 2);
        Settings.SeedInitialPrices = true;

        // Request BTCUSDT Crypto and Crypto Future data to form the cointegrating vectors and trade.
        _spot = AddCrypto("BTCUSDT", market: Market.Bybit).Symbol;
        _future = AddCryptoFuture("BTCUSDT", market: Market.Bybit).Symbol;

        // Daily recalibration on the cointegrating vectors.
        Schedule.On(
            DateRules.EveryDay(_future),
            TimeRules.At(0, 1),
            Recalibrate
        );
    }

    public override void OnData(Slice slice)
    {
        if (slice.Bars.TryGetValue(_future, out var y) && slice.Bars.TryGetValue(_spot, out var x1))
        {
```

```
        {
            // Calculate the spread based on the cointegration vector.
            var spread = CalculateSpread(y.Close, x1.Close, _cointegrationVector);
            var normalizedSpread = (spread - _residualMean) / _residualStd;

            // If the spread is too high, short the dependent and long the independent variables based on the
weightings of the cointegrating vectors.
            // Else, and vice versa.
            var totalWeights = Math.Abs(_cointegrationVector[1]) + 1m;
            if (normalizedSpread > _entryThreshold && !Portfolio[_future].IsShort)
            {
                SetHoldings(_future, -1m / totalWeights);
                SetHoldings(_spot, _cointegrationVector[1] / totalWeights);
            }
            else if (normalizedSpread < -_entryThreshold && !Portfolio[_future].IsLong)
            {
                SetHoldings(_future, 1m / totalWeights);
                SetHoldings(_spot, -_cointegrationVector[1] / totalWeights);
            }

            // Exit if spread is converged.
            if ((Portfolio[_future].IsShort && normalizedSpread < 0m) || (Portfolio[_future].IsLong &&
normalizedSpread > 0m))
            {
                Liquidate();
            }
        }
    }

    private void Recalibrate()
    {
        // Calculate the cointegration vector from historical data.
        var history = History<TradeBar>(new[] { _future, _spot }, 252, Resolution.Daily)
            .Where(x => x.ContainsKey(_future) && x.ContainsKey(_spot))
            .ToList();
        _cointegrationVector = CalculateCointegrationVector(history);
        var residual = history.Select(x => CalculateSpread(x[_future].Close, x[_spot].Close,
_cointegrationVector)).ToArray();
        _residualMean = residual.Average();
        _residualStd = Convert.ToDecimal(Math.Sqrt(residual.Sum(x => Math.Pow((double)(x - _residualMean), 2)) /
residual.Length));
    }

    private decimal[] CalculateCointegrationVector(IEnumerable<DataDictionary<TradeBar>> history)
    {
        // Use log price to eliminate the compounding effect.
        var y = history.Select(x => Math.Log((double)x[_future].Close)).ToArray();
        var x1 = history.Select(x => Math.Log((double)x[_spot].Close)).ToArray();

        // Create a matrix for the regression analysis.
        var n = new[] { y.Length, x1.Length }.Min();
        var designMatrix = new double[n][];
        for (int i = 0; i < n; i++)
        {
            designMatrix[i] = new[] { x1[i] };
        }

        try
        {
            // Perform regression using MathNet.Numerics
            var coefficients = MultipleRegression.QR(designMatrix, y.TakeLast(n).ToArray(), intercept: true);
            // The coefficients array will contain the intercept and the coefficients for each independent variable.
            return new[] { (decimal)coefficients[0], (decimal)coefficients[1] };
        }
        catch
        {
            return _cointegrationVector;
        }
    }

    private decimal CalculateSpread(decimal y, decimal x1, decimal[] cointegrationVector)
    {
        // Using the cointegration vector to calculate the spread.
        return LogPrice(y) - (cointegrationVector[0] + LogPrice(x1) * cointegrationVector[1]);
    }

    private decimal LogPrice(decimal price)
    {
        return Convert.ToDecimal(Math.Log((double)price));
    }
}
```

Asset Classes

Forex

Introduction

This page explains how to get historical data for Forex. Some of the data you can get include quotes, ticks, and indicator data.

Quotes

To get historical quote data , call the `History<QuoteBar>` method with a security's `Symbol` .

```
C#
public class ForexQuoteBarHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddForex("EURUSD").Symbol;
        // Get the 5 trailing minute QuoteBar objects of the security.
        var history = History<QuoteBar>(symbol, 5, Resolution.Minute);
        // Iterate through the QuoteBar objects and calculate the spread.
        foreach (var bar in history)
        {
            var t = bar.EndTime;
            var spread = bar.Ask.Close - bar.Bid.Close;
        }
    }
}
```

Ticks

To get historical tick data , call the `History<Tick>` method with a security's `Symbol` and `Resolution.Tick` .

```
C#
public class ForexTickHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddForex("EURUSD").Symbol;
        // Get the trailing 2 days of ticks for the security.
        var history = History<Tick>(symbol, TimeSpan.FromDays(2), Resolution.Tick);
        // Calculate the spread.
        var spread = history.Select(tick => tick.AskPrice - tick.BidPrice);
    }
}
```

Ticks are a sparse dataset, so request ticks over a trailing period of time or between start and end times.

Slices

To get historical Slice data, call the `History` method without passing any `Symbol` objects. This method returns `Slice` objects, which contain data points from all the datasets in your algorithm. If you omit the `resolution` argument, it uses the resolution that you set for each security and dataset when you created the subscriptions.

```C#
public class SliceHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);
        // Add some securities and datasets.
        AddForex("EURUSD");
        // Get the historical Slice objects over the last 5 minutes for all the subcriptions in your algorithm.
        var history = History(5, Resolution.Minute);
        // Iterate through each historical Slice.
        foreach (var slice in history)
        {
            // Iterate through each QuoteBar in this Slice.
            foreach (var kvp in slice.QuoteBars)
            {
                var symbol = kvp.Key;
                var bar = kvp.Value;
            }
        }
    }
}
```

Indicators

To get historical indicator values, call the `IndicatorHistory` method with an indicator and the security's `Symbol` .

```C#
public class ForexIndicatorHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddForex("EURUSD").Symbol;
        // Get the 21-day SMA values of the security for the last 5 trading days.
        var history = IndicatorHistory(new SimpleMovingAverage(21), symbol, 5, Resolution.Daily);
        // Get the maximum of the SMA values.
        var maxSMA = history.Max(indicatorDataPoint => indicatorDataPoint.Current.Value);
    }
}
```

The `IndicatorHistory` method resets your indicator, makes a history request, and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

To make the `IndicatorHistory` method update the indicator with an alternative price field instead of the close (or mid-price) of each bar, pass a `selector` argument.

```C#
// Get the historical values of an indicator over the last 30 days, applying the indicator to the security's ask
price.
var history = IndicatorHistory(indicator, symbol, TimeSpan.FromDays(30), selector: Field.AskClose);
```

Some indicators require the prices of two securities to compute their value (for example, Beta). In this case, pass a list of the `Symbol` objects to the method.

```
C#
public class ForexMultiAssetIndicatorHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Add the target and reference securities.
        var targetSymbol = AddForex("AUDUSD").Symbol;
        var referenceSymbol = AddForex("EURUSD").Symbol;
        // Create a 21-period Beta indicator.
        var beta = new Beta("", targetSymbol, referenceSymbol, 21);
        // Get the historical values of the indicator over the last 10 trading days.
        var history = IndicatorHistory(beta, new[] {targetSymbol, referenceSymbol}, 10, Resolution.Daily);
        // Get the average Beta value.
        var avgBeta = history.Average(indicatorDataPoint => indicatorDataPoint.Current.Value);
    }
}
```

If you already have a list of Slice objects, you can pass them to the `IndicatorHistory` method to avoid the internal history request.

```
C#
var slices = History(new[] {symbol}, 30, Resolution.Daily);
var history = IndicatorHistory(indicator, slices);
```

Examples

The following examples demonstrate some common practices for trading Forex with historical data.

Example 1: Arbitration with Cointegration

The below example uses linear regression with historical data of 3 related Forex pairs to obtain the cointegrating vector between them. We measure the spread between the 3 Forex pairs and trade mean reversal upon their spread divergence.

```
C#
using MathNet.Numerics.LinearRegression;

public class CointegrationVectorRegressionAlgorithm : QCAlgorithm
{
    private List<Symbol> _currencyPairs;
    private double[] _cointegrationVector = [0.0, 0.0, 0.0];
    private double _entryThreshold = 0.0001;
    private static readonly string[] sourceArray = ["USDJPY", "EURUSD", "EURJPY"];

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);
        Settings.SeedInitialPrices = true;

        // Request Forex data to form the cointegrating vectors and trade.
        _currencyPairs = sourceArray.Select(pair => AddForex(pair).Symbol).ToList();

        // Daily recalibration on the cointegrating vectors.
        Schedule.On(DateRules.EveryDay(_currencyPairs[0]), TimeRules.At(0, 1), Recalibrate);
    }

    public override void OnData(Slice slice)
    {
        if (slice.QuoteBars.TryGetValue(_currencyPairs[0], out var y) &&
slice.QuoteBars.TryGetValue(_currencyPairs[1], out var x1) && slice.QuoteBars.TryGetValue(_currencyPairs[2], out var
x2))
        {
            // Calculate the spread based on the cointegration vector.
            var spread = CalculateSpread(y.Close, x1.Close, x2.Close, _cointegrationVector);

            // Exit if spread is converged.
            if ((Portfolio[_currencyPairs[0]].IsShort && spread < 0) || (Portfolio[_currencyPairs[0]].IsLong &&
spread > 0))
```

```
spread > 0))
            {
                Liquidate();
                return;
            }

            // If spread is too high, short the dependent and long the independent variables based on the weightings
of the cointegrating vectors.
            // Else, vice versa.
            var totalWeights = 1 / (Math.Abs(_cointegrationVector[1]) + Math.Abs(_cointegrationVector[2]) + 1);
            if (spread > _entryThreshold)
            {
                SetHoldings(_currencyPairs[0], -totalWeights);
                SetHoldings(_currencyPairs[1], _cointegrationVector[1] * totalWeights);
                SetHoldings(_currencyPairs[2], _cointegrationVector[2] * totalWeights);
            }
            else if (spread < -_entryThreshold)
            {
                SetHoldings(_currencyPairs[0], totalWeights);
                SetHoldings(_currencyPairs[1], -_cointegrationVector[1] * totalWeights);
                SetHoldings(_currencyPairs[2], -_cointegrationVector[2] * totalWeights);
            }
        }
    }

    private void Recalibrate()
    {
        // Calculate the cointegration vector from historical data.
        var history = History<QuoteBar>(_currencyPairs, 252, Resolution.Daily);
        _cointegrationVector = CalculateCointegrationVector(history);
    }

    private double[] CalculateCointegrationVector(IEnumerable<DataDictionary<QuoteBar>> history)
    {
        // Use log price to eliminate the compounding effect.
        var y = history.Select(x => Math.Log((double)x[_currencyPairs[0]].Close)).ToArray();    // dependent
variable: USDJPY
        var x1 = history.Select(x => Math.Log((double)x[_currencyPairs[1]].Close)).ToArray();   // independent
variable 1: EURUSD
        var x2 = history.Select(x => Math.Log((double)x[_currencyPairs[2]].Close)).ToArray();   // independent
variable 2: EURJPY

        // Create a matrix for the regression analysis.
        var n = new[] { y.Length, x1.Length, x2.Length }.Min();
        var designMatrix = new double[n][];
        for (int i = 0; i < n; i++)
        {
            designMatrix[i] = [ x1[i], x2[i] ];
        }

        try
        {
            // Perform regression using MathNet.Numerics
            // The coefficients array will contain the intercept and the coefficients for each independent variable.
            // Format: [intercept, coeff_EURUSD, coeff_EURJPY]
            return MultipleRegression.QR(designMatrix, [.. y.TakeLast(n)], intercept: true);
        }
        catch
        {
            return _cointegrationVector;
        }
    }

    private static double CalculateSpread(decimal y, decimal x1, decimal x2, double[] cointegrationVector)
    {
        static double logPrice(decimal price) => Math.Log((double)price);
        // Using the cointegration vector to calculate the spread.
        return logPrice(y) - (cointegrationVector[0] + logPrice(x1) * cointegrationVector[1] + logPrice(x2) *
cointegrationVector[2]);
    }
}
```

Asset Classes

Futures

Introduction

This page explains how to get historical data for Futures. Some of the data you can get include prices, open interest, indicators, and contract chain data.

Trades

To get historical trade data , call the `History<TradeBar>` method with a security's `Symbol` .

```
C#
public class FutureTradeBarHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var future = AddFuture(Futures.Indices.SP500EMini);
        var symbol = FuturesChain(future.Symbol).First().Symbol;
        // Get the 5 trailing daily TradeBar objects of the security.
        var history = History<TradeBar>(symbol, 5, Resolution.Daily);
        // Iterate through each TradeBar and calculate its dollar volume.
        foreach (var bar in history)
        {
            var t = bar.EndTime;
            var dollarVolume = bar.Close * bar.Volume;
        }
    }
}
```

Quotes

To get historical quote data , call the `History<QuoteBar>` method with a security's `Symbol` .

```
C#
public class FutureQuoteBarHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var future = AddFuture(Futures.Indices.SP500EMini);
        var symbol = FuturesChain(future.Symbol).First().Symbol;
        // Get the 5 trailing minute QuoteBar objects of the security.
        var history = History<QuoteBar>(symbol, 5, Resolution.Minute);
        // Iterate through the QuoteBar objects and calculate the spread.
        foreach (var bar in history)
        {
            var t = bar.EndTime;
            var spread = bar.Ask.Close - bar.Bid.Close;
        }
    }
}
```

Ticks

To get historical tick data , call the `History<Tick>` method with a security's `Symbol` and `Resolution.Tick` .

```
C#
public class FutureTickHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var future = AddFuture(Futures.Indices.SP500EMini);
        var symbol = FuturesChain(future.Symbol).First().Symbol;
        // Get the trailing 2 days of ticks for the security.
        var history = History<Tick>(symbol, TimeSpan.FromDays(2), Resolution.Tick);
        // Select the ticks that represent trades, excluding the quote ticks.
        var trades = history.Where(tick => tick.TickType == TickType.Trade);
    }
}
```

Ticks are a sparse dataset, so request ticks over a trailing period of time or between start and end times.

Slices

To get historical Slice data, call the `History` method without passing any `Symbol` objects. This method returns `Slice` objects, which contain data points from all the datasets in your algorithm. If you omit the `resolution` argument, it uses the resolution that you set for each security and dataset when you created the subscriptions.

```
C#
public class SliceHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);
        // Add some securities and datasets.
        AddFuture(Futures.Indices.SP500EMini);
        // Get the historical Slice objects over the last 5 days for all the subcriptions in your algorithm.
        var history = History(5, Resolution.Daily);
        // Iterate through each historical Slice.
        foreach (var slice in history)
        {
            // Iterate through each TradeBar in this Slice.
            foreach (var kvp in slice.Bars)
            {
                var symbol = kvp.Key;
                var bar = kvp.Value;
            }
        }
    }
}
```

Continuous Contract Prices

To get historical trade, quote, or tick data for the continuous contract , use the `Symbol` property of the `Future` object when you make the history request.

```C#
public class FutureContinuousContractHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Add the Future and specify the contract rollover settings.
        var future = AddFuture(
            Futures.Indices.SP500EMini,
            dataMappingMode: DataMappingMode.OpenInterest,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            contractDepthOffset: 0
        );
        // Get the 5 trailing daily TradeBar objects of the continuous contract.
        var history = History<TradeBar>(future.Symbol, 5, Resolution.Daily);
        // Iterate through each TradeBar and calculate its dollar volume.
        foreach (var bar in history)
        {
            var t = bar.EndTime;
            var dollarVolume = bar.Close * bar.Volume;
        }
    }
}
```

Symbol Changes

To get historical symbol change data , call the `History<SymbolChangedEvent>` method with the continuous contract Symbol.

```C#
public class FutureSymbolChangeEventsHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Add the Future and specify the contract rollover settings.
        var future = AddFuture(
            Futures.Indices.SP500EMini,
            dataMappingMode: DataMappingMode.OpenInterest,
            dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
            contractDepthOffset: 0
        );
        // Get the contract rollovers for the trailing year.
        var history = History<SymbolChangedEvent>(future.Symbol, TimeSpan.FromDays(365));
        // Iterate through each rollover and select the Symbol of each contract.
        foreach (var symbolChangedEvent in history)
        {
            var t = symbolChangedEvent.EndTime;
            var oldSymbol = symbolChangedEvent.OldSymbol;
            var newSymbol = symbolChangedEvent.NewSymbol;
        }
    }
}
```

Open Interest

To get historical open interest data , call the `History<OpenInterest>` method with a security's `Symbol` .

```
C#
public class FutureOpenInterestHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var future = AddFuture(Futures.Indices.SP500EMini);
        var symbol = FuturesChain(future.Symbol).First().Symbol;
        // Get the 5 trailing daily OpenInterest objects of the security.
        var history = History<OpenInterest>(symbol, 5, Resolution.Daily);
        // Iterate through each OpenInterest and get it's value.
        foreach (var oi in history)
        {
            var t = oi.EndTime;
            var openInterest = oi.Value;
        }
    }
}
```

Contracts

To get historical Futures contracts, call the `History<FutureUniverse>` method with the continuous contract `Symbol` . The data this method returns contains information on all the currently tradable contracts, not just the contracts that pass your filter.

```
C#
public class FutureContractsHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Add a Future.
        var future = AddFuture(Futures.Indices.SP500EMini);
        // Get the daily contract data over the last two trading days.
        var history = History<FutureUniverse>(future.Symbol, 2);
        // Iterate through each day of the history.
        foreach (var futureUniverse in history)
        {
            var t = futureUniverse.EndTime;
            // Select the contract with the greatest open interest.
            var mostOi = futureUniverse.Select(c => c as FutureUniverse).OrderByDescending(c =>
c.OpenInterest).First();
        }
    }
}
```

Contango

To get historical contango data for VX Futures , call the `History< VIXCentralContango >` method with the dataset `Symbol` .

```C#
public class VIXContangoHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Add the dataset and save a reference to its Symbol.
        var symbol = AddData<VIXCentralContango>("VIX", Resolution.Daily).Symbol;
        //  Get the trailing 7 days of contango data
        var history = History<VIXCentralContango>(symbol, 7, Resolution.Daily);
        // Iterate each data point and access its attributes.
        foreach (var dataPoint in history)
        {
            var t = dataPoint.EndTime;
            var value = dataPoint.Contango_F2_Minus_F1;
        }
    }
}
```

Indicators

To get historical indicator values, call the `IndicatorHistory` method with an indicator and the security's `Symbol` .

```C#
public class FuturesIndicatorHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddFuture(Futures.Indices.SP500EMini).Symbol;
        // Get the 21-day SMA values of the security for the last 5 trading days.
        var history = IndicatorHistory(new SimpleMovingAverage(21), symbol, 5, Resolution.Daily);
        // Get the maximum of the SMA values.
        var maxSMA = history.Max(indicatorDataPoint => indicatorDataPoint.Current.Value);
    }
}
```

The `IndicatorHistory` method resets your indicator, makes a history request, and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

To make the `IndicatorHistory` method update the indicator with an alternative price field instead of the close (or mid-price) of each bar, pass a `selector` argument.

```C#
// Get the historical values of an indicator over the last 30 days, applying the indicator to the security's volume.
var history = IndicatorHistory(indicator, symbol, TimeSpan.FromDays(30), selector: Field.Volume);
```

Some indicators require the prices of two securities to compute their value (for example, Beta). In this case, pass a list of the `Symbol` objects to the method.

```C#
public class FuturesMultiAssetIndicatorHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Add the target and reference securities.
        var targetSymbol = AddFuture(Futures.Metals.Gold).Symbol;
        var referenceSymbol = AddFuture(Futures.Indices.SP500EMini).Symbol;
        // Create a 21-period Beta indicator.
        var beta = new Beta("", targetSymbol, referenceSymbol, 21);
        // Get the historical values of the indicator over the last 10 trading days.
        var history = IndicatorHistory(beta, new[] {targetSymbol, referenceSymbol}, 10, Resolution.Daily);
        // Get the average Beta value.
        var avgBeta = history.Average(indicatorDataPoint => indicatorDataPoint.Current.Value);
    }
}
```

If you already have a list of Slice objects, you can pass them to the `IndicatorHistory` method to avoid the internal history request.

```C#
var slices = History(new[] {symbol}, 30, Resolution.Daily);
var history = IndicatorHistory(indicator, slices);
```

Examples

The following examples demonstrate some common practices for trading Futures with historical data.

Example 1: Stock-Gold All-Weather Portfolio

The following algorithm constructs a monthly-rebalanced, risk-parity portfolio using S&P 500 E-Mini and Micro Gold Futures contracts. A risk-parity approach balances the contribution to risk from each asset class, helping reduce overall portfolio drawdowns and improve diversification. The position sizing is determined using 1-year of historical daily returns for the selected Futures universe.

C#

```
public class FutureHistoricalDataAlgorithm : QCAlgorithm
{
    private List<Future> _futures;
    private IPortfolioOptimizer _optimizer;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1_000_000);
        Settings.SeedInitialPrices = true;

        // Select the future contracts representing various asset classes to diversify the portfolio.
        _futures = [.. new[] { Futures.Indices.SP500EMini, Futures.Metals.MicroGold }
            .Select(x => AddFuture(x,
                fillForward: true,
                extendedMarketHours: true,
                dataNormalizationMode: DataNormalizationMode.BackwardsRatio,
                dataMappingMode: DataMappingMode.OpenInterest,
                contractDepthOffset: 0))];

        // Instantiate the optimizer to perform risk-parity optimization.
        // Risk Parity portfolio can dissipate the risk of each crypto pair to reduce overall drawdown.
        _optimizer = new RiskParityPortfolioOptimizer(0, 1);

        // Set a scheduled event to rebalance the portfolio at the start of every month.
        Schedule.On(
            DateRules.MonthStart(),
            TimeRules.AfterMarketOpen(_futures[0].Symbol, 1),
            Rebalance
        );

        SetWarmUp(1);
    }

    private void Rebalance()
    {
        // Historical data request to get 1-year data for optimization.
        var symbols = _futures.Select(x => x.Symbol).ToList();
        var history = History<TradeBar>(symbols, 253, Resolution.Daily)
            .Where(x => _futures.All(y => x.ContainsKey(y.Symbol)))
            .ToList();
        // Daily return on the universe members to calculate the optimized weights.
        var returns = GetReturns(history, symbols);

        // Calculate the optimized weights.
        var weights = _optimizer.Optimize(returns);

        // Rebalance the portfolio according to the optimized weights.
        var targets = Enumerable.Range(0, weights.Length)
            .Select(i => new PortfolioTarget(_futures[i].Mapped, Convert.ToDecimal(weights[i])/2))
            .ToList();
        SetHoldings(targets, liquidateExistingHoldings: true);
    }

    private double[,] GetReturns(List<DataDictionary<TradeBar>> history, List<Symbol> symbols)
    {
        // Create a 2d array of historical daily returns from historical price data.
        var returns = new double[history.Count, symbols.Count];
        for (int j = 0; j < symbols.Count; j++)
        {
            var lastPrice = 0m;
            for (int i = 0; i < history.Count; i++)
            {
                var current = history[i][symbols[j]].Close;
                if (i > 0)
                {
                    returns[i, j] = (double)((current - lastPrice) / lastPrice);
                }
                lastPrice = current;
            }
        }
        return returns;
    }
}
```

Asset Classes

Future Options

Introduction

This page explains how to get historical data for Future Options. Some of the data you can get include prices and indicator data.

Trades

To get historical trade data , call the `History<TradeBar>` method with a security's `Symbol` .

```C#
public class FutureOptionsTradeBarHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Add a FOP universe.
        var future = AddFuture(Futures.Indices.SP500EMini);
        future.SetFilter(universe => universe.FrontMonth());
        AddFutureOption(future.Symbol, universe => universe.FrontMonth().Strikes(-1, 0).CallsOnly());
    }

    // Get trailing data whenever a new FOP contract enters the universe.
    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            if (security.Type == SecurityType.FutureOption)
            {
                // Get the 3 trailing daily TradeBar objects of the security.
                var history = History<TradeBar>(security.Symbol, 3, Resolution.Daily);
                // Iterate through each TradeBar and calculate its dollar volume.
                foreach (var bar in history)
                {
                    var t = bar.EndTime;
                    var dollarVolume = bar.Close * bar.Volume;
                }
            }
        }
    }
}
```

Request minute, hour, or daily resolution data. Otherwise, the history request won't return any data.

Quotes

To get historical quote data , call the `History<QuoteBar>` method with a security's `Symbol` .

```
C#
public class FutureOptionsQuoteBarHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Add a FOP universe.
        var future = AddFuture(Futures.Indices.SP500EMini);
        future.SetFilter(universe => universe.FrontMonth());
        AddFutureOption(future.Symbol, universe => universe.FrontMonth().Strikes(-1, 0).CallsOnly());
    }

    // Get trailing data whenever a new FOP contract enters the universe.
    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            if (security.Type == SecurityType.FutureOption)
            {
                // Get the 3 trailing daily QuoteBar objects of the security.
                var history = History<QuoteBar>(security.Symbol, 3, Resolution.Daily);
                // Iterate through the QuoteBar objects and calculate the spread.
                foreach (var bar in history)
                {
                    var t = bar.EndTime;
                    var spread = bar.Ask.Close - bar.Bid.Close;
                }
            }
        }
    }
}
```

Request minute, hour, or daily resolution data. Otherwise, the history request won't return any data.

Slices

To get historical Slice data, call the `History` method without passing any `Symbol` objects. This method returns `Slice` objects, which contain data points from all the datasets in your algorithm. If you omit the `resolution` argument, it uses the resolution that you set for each security and dataset when you created the subscriptions.

```
C#
public class SliceHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Add some securities and datasets.
        var future = AddFuture(Futures.Indices.SP500EMini);
        future.SetFilter(universe => universe.FrontMonth());
        AddFutureOption(future.Symbol, universe => universe.FrontMonth().Strikes(-1, 0).CallsOnly());
        // Add a Scheduled Event that runs at the start of each month.
        Schedule.On(DateRules.MonthStart(future.Symbol), TimeRules.AfterMarketOpen(future.Symbol, 60), Trade);
    }

    public void Trade()
    {
        // Get the historical Slice objects over the last 30 minutes for all the subcriptions in your algorithm.
        var history = History(30, Resolution.Minute);
        // Iterate through each historical Slice.
        foreach (var slice in history)
        {
            // Iterate through each TradeBar in this Slice.
            foreach (var kvp in slice.Bars)
            {
                var symbol = kvp.Key;
                var bar = kvp.Value;
            }
        }
    }
}
```

Indicators

To get historical indicator values, call the `IndicatorHistory` method with an indicator and the security's `Symbol` .

```
C#
public class FutureOptionIndicatorHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Add a FOP universe.
        var future = AddFuture(Futures.Indices.SP500EMini);
        future.SetFilter(universe => universe.FrontMonth());
        AddFutureOption(future.Symbol, universe => universe.FrontMonth().Strikes(0, 0).CallsOnly());
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            if (security.Type != SecurityType.FutureOption)
            {
                continue;
            }
            // Get the 21-day SMA values of the contract for the last 5 trading days.
            var history = IndicatorHistory(new SimpleMovingAverage(21), security.Symbol, 5, Resolution.Daily);
            // Get the maximum of the SMA values.
            var maxSMA = history.Max(indicatorDataPoint => indicatorDataPoint.Current.Value);
        }
    }
}
```

The `IndicatorHistory` method resets your indicator, makes a history request, and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

To make the `IndicatorHistory` method update the indicator with an alternative price field instead of the close (or mid-price) of each bar, pass a `selector` argument.

```
C#
// Get the historical values of an indicator over the last 30 days, applying the indicator to the contract's volume.
var history = IndicatorHistory(indicator, symbol, TimeSpan.FromDays(30), selector: Field.Volume);
```

Some indicators require the prices of multiple securities to compute their value (for example, the indicators for the Greeks and implied volatility). In this case, pass a list of the `Symbol` objects to the method.

```C#
public class FutureOptionMultiAssetIndicatorHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Add a FOP universe.
        var future = AddFuture(Futures.Indices.SP500EMini);
        future.SetFilter(universe => universe.FrontMonth());
        AddFutureOption(future.Symbol, universe => universe.FrontMonth().Strikes(0, 0).CallsOnly());
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            if (security.Type != SecurityType.FutureOption)
            {
                continue;
            }
            var option = security.Symbol;
            // Get the Symbol of the mirror contract.
            var mirror = QuantConnect.Symbol.CreateOption(
                option.Underlying, option.ID.Market, option.ID.OptionStyle,
                option.ID.OptionRight == OptionRight.Put ? OptionRight.Call : OptionRight.Put,
                option.ID.StrikePrice, option.ID.Date
            );
            // Create the indicator.
            var indicator = new ImpliedVolatility(option, RiskFreeInterestRateModel, new
ConstantDividendYieldModel(0), mirror, OptionPricingModelType.ForwardTree);
            // Get the historical values of the indicator over the last 10 trading minutes.
            var history = IndicatorHistory(indicator, new[] {option.Underlying, option, mirror}, 10,
Resolution.Minute);
            // Get the average IV value.
            var avgIV = history.Average(indicatorDataPoint => indicatorDataPoint.Current.Value);
        }
    }
}
```

If you already have a list of Slice objects, you can pass them to the `IndicatorHistory` method to avoid the internal history request.

```C#
var slices = History(new[] {symbol}, 30, Resolution.Daily);
var history = IndicatorHistory(indicator, slices);
```

Examples

The following examples demonstrate some common practices for trading Future Options with historical data.

Example 1: Trend Following on Front Month ATM Option Contract

This algorithm strategically trades the front month S&P 500 EMini Future Options by analyzing bid and ask volumes shortly after the market opens. Using scheduled events, it effectively executes trades based on historical quote data, optimizing decision-making. The algorithm aims for timely entries and exits, ensuring efficient capital management and quick adaptability in volatile markets.

C#

```
public class FutureOptionsTradingAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(200000);
        Settings.SeedInitialPrices = true;

        // Request ES option data for trading and signal generation.
        var future = AddFuture(Futures.Indices.SP500EMini);
        future.SetFilter(universe => universe.FrontMonth());
        // We are interested in ATM front-month options since they are the most popular.
        AddFutureOption(future.Symbol, universe => universe.Expiration(0, 90).Strikes(-3, 3));

        // Schedule event to enter and exit option contract position.
        Schedule.On(DateRules.EveryDay(future.Symbol), TimeRules.AfterMarketOpen(future.Symbol, 16), TradeOption);
        Schedule.On(DateRules.EveryDay(future.Symbol), TimeRules.BeforeMarketClose(future.Symbol, 15), () =>
Liquidate());
    }

    public void TradeOption()
    {
        // Get the option chain to trade.
        foreach (var chain in CurrentSlice.OptionChains.Values)
        {
            foreach (var option in chain)
            {
                // Request historical quote data for signal generation.
                var history = History<QuoteBar>(option.Symbol, 15, Resolution.Minute);
                if (history.Any()){
                    // Calculate the total bid and ask for dollar volume to determine the capital directional force.
                    var totalBidVolume = 0m;
                    var totalAskVolume = 0m;
                    foreach (var bar in history)
                    {
                        if (bar.Bid != null)
                        {
                            totalBidVolume += bar.Bid.Close * bar.LastBidSize;
                        }
                        if (bar.Ask != null)
                        {
                            totalAskVolume += bar.Ask.Close * bar.LastAskSize;
                        }
                    }

                    // Follow the capital flow to trade.
                    if (totalBidVolume > totalAskVolume)
                    {
                        MarketOrder(option.Symbol, 1);
                    }
                    else
                    {
                        MarketOrder(option.Symbol, -1);
                    }
                }
            }

        }
    }
}
```

Asset Classes

Index

Introduction

This page explains how to get historical price and indicator data for Indices.

Bars

To get historical price data , call the `History` method with an Index `Symbol` .

```C#
public class IndexPriceHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of an Index.
        var symbol = AddIndex("SPX").Symbol;
        // Get the 5 trailing daily bars of the Index.
        var history = History(symbol, 5, Resolution.Daily);
        // Iterate through each bar and get its price.
        foreach (var bar in history)
        {
            var t = bar.EndTime;
            var price = bar.Close;
        }
    }
}
```

Request minute, hour, or daily resolution data. Otherwise, the history request won't return any data.

Slices

To get historical Slice data, call the `History` method without passing any `Symbol` objects. This method returns `Slice` objects, which contain data points from all the datasets in your algorithm. If you omit the `resolution` argument, it uses the resolution that you set for each security and dataset when you created the subscriptions.

```C#
public class SliceHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);
        // Add some securities and datasets.
        AddIndex("SPX");
        // Get the historical Slice objects over the last 5 days for all the subcriptions in your algorithm.
        var history = History(5, Resolution.Daily);
        // Iterate through each historical Slice.
        foreach (var slice in history)
        {
            // Iterate through each TradeBar in this Slice.
            foreach (var kvp in slice.Bars)
            {
                var symbol = kvp.Key;
                var bar = kvp.Value;
            }
        }
    }
}
```

Indicators

To get historical indicator values, call the `IndicatorHistory` method with an indicator and the security's `Symbol` .

```
C#
public class IndexIndicatorHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddIndex("SPX").Symbol;
        // Get the 21-day SMA values of the security for the last 5 trading days.
        var history = IndicatorHistory(new SimpleMovingAverage(21), symbol, 5, Resolution.Daily);
        // Get the maximum of the SMA values.
        var maxSMA = history.Max(indicatorDataPoint => indicatorDataPoint.Current.Value);
    }
}
```

	current	**rollingsum**
2024-12-12 15:15:00	6003.055714	126064.17
2024-12-13 15:15:00	6006.096667	126128.03
2024-12-16 15:15:00	6012.036190	126252.76
2024-12-17 15:15:00	6020.634286	126433.32
2024-12-18 15:15:00	6019.442381	126408.29

The `IndicatorHistory` method resets your indicator, makes a history request, and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

To make the `IndicatorHistory` method update the indicator with an alternative price field instead of the close (or mid-price) of each bar, pass a `selector` argument.

```
C#
// Get the historical values of an indicator over the last 30 days, applying the indicator to the security's high
price.
var history = IndicatorHistory(indicator, symbol, TimeSpan.FromDays(30), selector: Field.High);
```

Some indicators require the prices of two securities to compute their value (for example, Beta). In this case, pass a list of the `Symbol` objects to the method.

```C#
public class IndexMultiAssetIndicatorHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Add the target and reference securities.
        var targetSymbol = AddIndex("NDX").Symbol;
        var referenceSymbol = AddIndex("SPX").Symbol;
        // Create a 21-period Beta indicator.
        var beta = new Beta("", targetSymbol, referenceSymbol, 21);
        // Get the historical values of the indicator over the last 10 trading days.
        var history = IndicatorHistory(beta, new[] {targetSymbol, referenceSymbol}, 10, Resolution.Daily);
        // Get the average Beta value.
        var avgBeta = history.Average(indicatorDataPoint => indicatorDataPoint.Current.Value);
    }
}
```

If you already have a list of Slice objects, you can pass them to the `IndicatorHistory` method to avoid the internal history request.

```C#
var slices = History(new[] {symbol}, 30, Resolution.Daily);
var history = IndicatorHistory(indicator, slices);
```

Examples

The following examples demonstrate some common practices for trading with Index historical data.

Example 1: Replicating Portfolio

The example below constructed a replicating SGX portfolio using the top 100 weighted SPY constituents. It provides an alternative way for retail investors to invest in SGX without access to certain financial instruments and a hedging idea on SGX close hours.

```C#
using MathNet.Numerics.LinearRegression;

public class SGXMatchingTradingAlgorithm : QCAlgorithm
{
    private Symbol _sgx;
    private Universe _universe;
    private decimal[] _weights = null;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        SetWarmUp(7, Resolution.Daily);

        // Request SGX Index data for matching.
        _sgx = AddIndex("SGX").Symbol;

        // We will try to use SPY constituents to reproduce the SGX index.
        _universe = AddUniverse(Universe.ETF("SPY", UniverseSettings, constituents =>
        {
            return constituents
                .Where(x => x.Weight.HasValue)
                .OrderByDescending(x => x.Weight.Value)
                .Take(100)
                .Select(x => x.Symbol);
        }));

        // Schedule an event to rebalance the portfolio.
        Schedule.On(
            DateRules.EveryDay("SPY"),
            TimeRules.AfterMarketOpen("SPY", 1),
            Rebalance
```

```
        Rebalance
    );
}

private void Rebalance()
{
    if (IsWarmingUp || _universe.Selected.Count == 0)
    {
        return;
    }

    var symbols = _universe.Selected.ToList();

    // Obtain the historical data.
    var allSymbols = new List<Symbol> { _sgx };
    allSymbols.AddRange(symbols);
    var historyRequest = History<TradeBar>(allSymbols, 252, Resolution.Daily).ToList();

    if (historyRequest.Count == 0)
    {
        return;
    }

    // Organize data by date.
    var dataByDate = new Dictionary<DateTime, Dictionary<Symbol, TradeBar>>();
    foreach (var slice in historyRequest)
    {
        foreach (var kvp in slice)
        {
            var symbol = kvp.Key;
            var bar = kvp.Value;
            var date = bar.Time.Date;

            if (!dataByDate.TryGetValue(date, out Dictionary<Symbol, TradeBar> value))
            {
                value = [];
                dataByDate[date] = value;
            }

            value[symbol] = bar;
        }
    }

    // Filter to dates where we have all symbols.
    var completeDates = dataByDate
        .Where(kvp => kvp.Value.ContainsKey(_sgx) && symbols.All(s => kvp.Value.ContainsKey(s)))
        .OrderBy(kvp => kvp.Key)
        .Select(kvp => kvp.Value)
        .ToList();

    if (completeDates.Count < 50)
    {
        return;
    }

    var benchmark = completeDates.Select(d => (double)d[_sgx].Close).ToArray();

    var designMatrix = new double[completeDates.Count][];
    for (int i = 0; i < completeDates.Count; i++)
    {
        designMatrix[i] = [.. symbols.Select(s => (double)completeDates[i][s].Close)];
    }

    CalculateWeights(designMatrix, benchmark);

    if (_weights != null)
    {
        var targets = _weights
            .Zip(symbols, (weight, symbol) => new PortfolioTarget(symbol, weight))
            .ToList();
        SetHoldings(targets, liquidateExistingHoldings: true);
    }
}

private void CalculateWeights(double[][] history, double[] benchmark)
{
    try
    {
        // Use log price to eliminate the compounding effect.
        var y = benchmark.Select(x => Math.Log(x)).ToArray();
        var designMatrix = history.Select(row => row.Select(x => Math.Log(x)).ToArray()).ToArray();
```

```
            // Perform regression to obtain the best-replicating portfolio of SGX.
            var coefficients = MultipleRegression.QR(designMatrix, y, intercept: false);

            // The coefficients array will contain the coefficients for each constituents.
            var sum = coefficients.Sum(c => Math.Abs(c));
            if (sum > 0)
            {
                _weights = [.. coefficients.Select(c => Convert.ToDecimal(c / sum))];
            }
        }
        catch
        {
        }
    }
}
```

Asset Classes

Index Options

Introduction

This page explains how to get historical data for Index Options. Some of the data you can get include prices, open interest, Greeks, and universe selection data.

Trades

To get historical trade data , call the `History<TradeBar>` method with a security's `Symbol` .

```C#
public class IndexOptionTradeBarHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var index = AddIndex("SPX");
        var symbol = OptionChain(index.Symbol).OrderBy(c => c.OpenInterest).Last().Symbol;
        // Get the 5 trailing daily TradeBar objects of the security.
        var history = History<TradeBar>(symbol, 5, Resolution.Daily);
        // Iterate through each TradeBar and calculate its dollar volume.
        foreach (var bar in history)
        {
            var t = bar.EndTime;
            var dollarVolume = bar.Close * bar.Volume;
        }
    }
}
```

Request minute, hour, or daily resolution data. Otherwise, the history request won't return any data.

Quotes

To get historical quote data , call the `History<QuoteBar>` method with a security's `Symbol` .

```C#
public class IndexOptionQuoteBarHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var index = AddIndex("SPX");
        var symbol = OptionChain(index.Symbol).OrderBy(c => c.OpenInterest).Last().Symbol;
        // Get the 5 trailing minute QuoteBar objects of the security.
        var history = History<QuoteBar>(symbol, 5, Resolution.Minute);
        // Iterate through the QuoteBar objects and calculate the spread.
        foreach (var bar in history)
        {
            var t = bar.EndTime;
            var spread = bar.Ask.Close - bar.Bid.Close;
        }
    }
}
```

Request minute, hour, or daily resolution data. Otherwise, the history request won't return any data.

Slices

To get historical Slice data, call the `History` method without passing any `Symbol` objects. This method returns `Slice` objects, which contain data points from all the datasets in your algorithm. If you omit the `resolution` argument, it uses the resolution that you set for each security and dataset when you created the subscriptions.

```
C#
public class SliceHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);
        // Add some securities and datasets.
        var index = AddIndex("SPX");
        var contract = OptionChain(index.Symbol).OrderBy(c => c.OpenInterest).Last();
        AddOptionContract(contract.Symbol);
        // Get the historical Slice objects over the last 5 days for all the subcriptions in your algorithm.
        var history = History(5, Resolution.Daily);
        // Iterate through each historical Slice.
        foreach (var slice in history)
        {
            // Iterate through each TradeBar in this Slice.
            foreach (var kvp in slice.Bars)
            {
                var symbol = kvp.Key;
                var bar = kvp.Value;
            }
        }
    }
}
```

Open Interest

To get historical open interest data , call the `History<OpenInterest>` method with a security's `Symbol` .

```
C#
public class IndexOptionOpenInterestHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var index = AddIndex("SPX");
        var symbol = OptionChain(index.Symbol).OrderBy(c => c.OpenInterest).Last().Symbol;
        // Get the 5 trailing daily OpenInterest objects of the security.
        var history = History<OpenInterest>(symbol, 5, Resolution.Daily);
        // Iterate through each OpenInterest and get it's value.
        foreach (var oi in history)
        {
            var t = oi.EndTime;
            var openInterest = oi.Value;
        }
    }
}
```

Daily Option Chains

To get historical daily Option chain data , call the `History` method with the Option `Symbol` object. The data this method returns contains information on all the currently tradable contracts, not just the contracts that pass your filter.

```C#
public class IndexOptionDailyOptionChainHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 23);
        SetEndDate(2024, 12, 31);
        // Add an Index Option universe.
        var option = AddIndexOption("SPX");
        // Get the trailing 5 daily Option chains.
        var history = History<OptionUniverse>(option.Symbol, 5);
        // Iterate through each day of the history.
        foreach (var optionUniverse in history)
        {
            var t = optionUniverse.EndTime;
            // Select the 2 contracts with the most volume.
            var mostTraded = optionUniverse.Select(c => c as OptionUniverse).OrderByDescending(c =>
c.Volume).Take(2);
        }
    }
}
```

Indicators

To get historical indicator values, call the `IndicatorHistory` method with an indicator and the security's `Symbol` .

```C#
public class IndexOptionIndicatorHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of the Option contract.
        var underlying = AddIndex("SPX").Symbol;
        var symbol = OptionChain(underlying).OrderBy(c => c.OpenInterest).Last().Symbol;
        // Get the 21-day SMA values of the contract for the last 5 trading days.
        var history = IndicatorHistory(new SimpleMovingAverage(21), symbol, 5, Resolution.Daily);
        // Get the maximum of the SMA values.
        var maxSMA = history.Max(indicatorDataPoint => indicatorDataPoint.Current.Value);
    }
}
```

The `IndicatorHistory` method resets your indicator, makes a history request, and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

To make the `IndicatorHistory` method update the indicator with an alternative price field instead of the close (or mid-price) of each bar, pass a `selector` argument.

```C#
// Get the historical values of an indicator over the last 30 days, applying the indicator to the security's volume.
var history = IndicatorHistory(indicator, symbol, TimeSpan.FromDays(30), selector: Field.Volume);
```

Some indicators require the prices of multiple securities to compute their value (for example, the indicators for the Greeks and implied volatility). In this case, pass a list of the `Symbol` objects to the method.

```C#
public class IndexOptionMultiAssetIndicatorHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of the underlying asset.
        var underlying = AddIndex("SPX").Symbol;
        // Get the Option contract Symbol.
        var option = OptionChain(underlying).OrderBy(c => c.OpenInterest).Last().Symbol;
        // Get the Symbol of the mirror contract.
        var mirror = QuantConnect.Symbol.CreateOption(
            underlying, option.ID.Market, option.ID.OptionStyle,
            option.ID.OptionRight == OptionRight.Put ? OptionRight.Call : OptionRight.Put,
            option.ID.StrikePrice, option.ID.Date
        );
        // Create the indicator.
        var indicator = new ImpliedVolatility(option, RiskFreeInterestRateModel, new
DividendYieldProvider(underlying), mirror, OptionPricingModelType.ForwardTree);
        // Get the historical values of the indicator over the last 60 trading minutes.
        var history = IndicatorHistory(indicator, new[] {underlying, option, mirror}, 60, Resolution.Minute);
        // Get the average IV value.
        var avgIV = history.Average(indicatorDataPoint => indicatorDataPoint.Current.Value);
    }
}
```

If you already have a list of Slice objects, you can pass them to the `IndicatorHistory` method to avoid the internal history request.

```C#
var slices = History(new[] {symbol}, 30, Resolution.Daily);
var history = IndicatorHistory(indicator, slices);
```

Examples

The following examples demonstrate some common practices for trading Index Options with historical data.

Example 1: Standard-Weekly Contracts Cointegration

The following example analyzes the cointegration relationship between the front-month ATM SPX and SPXW calls. By measuring their spread divergence, we trade mean reversal on their spread convergence.

```C#
using MathNet.Numerics.LinearRegression;

public class IndexOptionHistoricalDataAlgorithm : QCAlgorithm
{
    private Symbol _spxContract, _spxwContract;
    private decimal[] _cointegrationVector = new[] { 0m, 0m };
    private decimal _threshold = 2m;
    private decimal _spreadMean = 0m;
    private decimal _spreadSd = 1m;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(10000000);

        // Select the Index Options to analyze and trade by week.
        Schedule.On(
            DateRules.WeekStart(),
            TimeRules.At(9, 15),
            SelectContract
        );
    }

    public override void OnData(Slice slice)
    {
        if (_spxwContract != null && slice.QuoteBars.TryGetValue(_spxwContract, out var x) &&
```

```
        if (_spxwContract != null && slice.QuoteBars.TryGetValue(_spxwContract, out var y) &&
slice.QuoteBars.TryGetValue(_spxContract, out var x1))
        {
            // Obtain the current spread to see if there is any price divergence to trade.
            var spread = CalculateSpread(y.Close, x1.Close, _cointegrationVector);
            var z = (spread - _spreadMean) / _spreadSd;

            // If the spread diverges above or below the threshold, trade to bet on mean reversal.
            if (z > _threshold && !Portfolio[_spxContract].IsLong)
            {
                MarketOrder(_spxContract, 10 * _cointegrationVector[1]);
                MarketOrder(_spxwContract, -10);
            }
            else if (z < -_threshold && !Portfolio[_spxwContract].IsLong)
            {
                MarketOrder(_spxContract, -10 * _cointegrationVector[1]);
                MarketOrder(_spxwContract, 10);
            }

            // If prices converge, exit positions.
            if ((Portfolio[_spxContract].IsLong && z < 0m) || (Portfolio[_spxwContract].IsLong && z > 0m))
            {
                Liquidate();
            }
        }
    }

    private void SelectContract()
    {
        var index = QuantConnect.Symbol.Create("SPX", SecurityType.Index, Market.USA);
        // Obtain the SPX ATM call contract since it is the most liquid with which to trade.
        var spx = QuantConnect.Symbol.CreateCanonicalOption(index);
        var spxContracts = OptionChain(spx).Where(x => x.Expiry <= Time.AddDays(30)).ToList();
        if (spxContracts.Count == 0)
        {
            _spxContract = null;
            _spxwContract = null;
            return;
        }
        var expiry = spxContracts.Max(x => x.Expiry);
        _spxContract = spxContracts.Where(x => x.Expiry == expiry && x.Right == OptionRight.Call)
            .OrderBy(x => Math.Abs(x.Strike - x.UnderlyingLastPrice))
            .First().Symbol;
        // Obtain the SPXW contract with the same strike, right, and expiry.
        var spxw = QuantConnect.Symbol.CreateCanonicalOption(index, "SPXW", Market.USA, "?SPXW");
        _spxwContract = OptionChain(spxw).Where(x => x.Expiry == expiry && x.Right == OptionRight.Call && x.Strike ==
_spxContract.ID.StrikePrice).FirstOrDefault()?.Symbol;

        if (_spxwContract != null)
        {
            // Subscribe to the contracts we will trade
            AddIndexOptionContract(_spxContract);
            AddIndexOptionContract(_spxwContract);
            // Obtain the historical data and find their cointegration relationship.
            var history = History<QuoteBar>(new[] { _spxContract, _spxwContract }, 1000, Resolution.Minute).Where(x
=> x.ContainsKey(_spxContract) && x.ContainsKey(_spxwContract)).ToList();
            _cointegrationVector = CalculateCointegrationVector(history);
            // Obtain the mean and SD of the spread between the options.
            var residual = history.Select(x => CalculateSpread(x[_spxwContract].Close, x[_spxContract].Close,
_cointegrationVector)).ToList();
            _spreadMean = residual.Average();
            _spreadSd = Convert.ToDecimal(Math.Sqrt(residual.Sum(x => Math.Pow((double)(x - _spreadMean), 2) /
residual.Count)));
        }
    }

    private decimal[] CalculateCointegrationVector(IEnumerable<DataDictionary<QuoteBar>> history)
    {
        // Use log price to eliminate the compounding effect.
        var y = history.Select(x => Math.Log((double)x[_spxwContract].Close)).ToArray();
        var x1 = history.Select(x => Math.Log((double)x[_spxContract].Close)).ToArray();

        // Create a matrix for the regression analysis.
        var n = new[] { y.Length, x1.Length }.Min();
        var designMatrix = new double[n][];
        for (int i = 0; i < n; i++)
        {
            designMatrix[i] = new[] { x1[i] };
        }

        try
        {
```

```
        {
            // Perform regression using MathNet.Numerics
            var coefficients = MultipleRegression.QR(designMatrix, y.TakeLast(n).ToArray(), intercept: true);

            // The coefficients array will contain the intercept and the coefficients for each independent variable.
            // Format: [intercept, x1]
            return new[] { (decimal)coefficients[0], (decimal)coefficients[1] };
        }
        catch
        {
            return _cointegrationVector;
        }
    }

    private decimal CalculateSpread(decimal y, decimal x1, decimal[] cointegrationVector)
    {
        // Using the cointegration vector to calculate the spread.
        return LogPrice(y) - (cointegrationVector[0] + LogPrice(x1) * cointegrationVector[1]);
    }

    private decimal LogPrice(decimal price)
    {
        return Convert.ToDecimal(Math.Log((double)price));
    }
}
```

Asset Classes

CFD

Introduction

This page explains how to get historical quotes and indicator data for CFDs.

Quotes

To get historical quote data , call the `History<QuoteBar>` method with a security's `Symbol` .

```C#
public class CFDQuoteBarHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddCfd("XAUUSD").Symbol;
        // Get the 5 trailing minute QuoteBar objects of the security.
        var history = History<QuoteBar>(symbol, 5, Resolution.Minute);
        // Iterate through the QuoteBar objects and calculate the spread.
        foreach (var bar in history)
        {
            var t = bar.EndTime;
            var spread = bar.Ask.Close - bar.Bid.Close;
        }
    }
}
```

		askclose	askhigh	asklow	askopen	bidclose	bidhigh	bidlow	bidopen	close	high	low	open
symbol	**time**												
XAUUSD	**2024-12-18 23:56:00**	2607.96	2608.53	2607.91	2608.53	2607.49	2607.99	2607.43	2607.99	2607.725	2608.26	2607.67	2608.26
	2024-12-18 23:57:00	2608.31	2608.36	2607.92	2607.96	2607.74	2607.80	2607.45	2607.49	2608.025	2608.08	2607.685	2607.72
	2024-12-18 23:58:00	2608.47	2608.55	2608.28	2608.31	2607.89	2608.01	2607.71	2607.74	2608.180	2608.28	2607.995	2608.02
	2024-12-18 23:59:00	2609.48	2609.70	2608.43	2608.47	2609.05	2609.12	2607.89	2607.89	2609.265	2609.41	2608.160	2608.18
	2024-12-19 00:00:00	2609.48	2610.17	2609.48	2609.48	2609.10	2609.65	2609.03	2609.05	2609.290	2609.91	2609.255	2609.26

Ticks

To get historical tick data , call the `History<Tick>` method with a security's `Symbol` and `Resolution.Tick` .

```
C#
public class CFDTickHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddCfd("XAUUSD").Symbol;
        // Get the trailing 2 days of ticks for the security.
        var history = History<Tick>(symbol, TimeSpan.FromDays(2), Resolution.Tick);
        // Calculate the spread.
        var spread = history.Select(tick => tick.AskPrice - tick.BidPrice);
    }
}
```

Ticks are a sparse dataset, so request ticks over a trailing period of time or between start and end times.

Slices

To get historical Slice data, call the `History` method without passing any `Symbol` objects. This method returns `Slice` objects, which contain data points from all the datasets in your algorithm. If you omit the `resolution` argument, it uses the resolution that you set for each security and dataset when you created the subscriptions.

```
C#
public class SliceHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);
        // Add some securities and datasets.
        AddCfd("XAUUSD");
        // Get the historical Slice objects over the last 5 minutes for all the subcriptions in your algorithm.
        var history = History(5, Resolution.Minute);
        // Iterate through each historical Slice.
        foreach (var slice in history)
        {
            // Iterate through each QuoteBar in this Slice.
            foreach (var kvp in slice.QuoteBars)
            {
                var symbol = kvp.Key;
                var bar = kvp.Value;
            }
        }
    }
}
```

Indicators

To get historical indicator values, call the `IndicatorHistory` method with an indicator and the security's `Symbol` .

```C#
public class CFDIndicatorHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a security.
        var symbol = AddCfd("SPX500USD").Symbol;
        // Get the 21-day SMA values of the security for the last 5 trading days.
        var history = IndicatorHistory(new SimpleMovingAverage(21), symbol, 5, Resolution.Daily);
        // Get the maximum of the SMA values.
        var maxSMA = history.Max(indicatorDataPoint => indicatorDataPoint.Current.Value);
    }
}
```

The `IndicatorHistory` method resets your indicator, makes a history request, and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

To make the `IndicatorHistory` method update the indicator with an alternative price field instead of the close (or mid-price) of each bar, pass a `selector` argument.

```C#
// Get the historical values of an indicator over the last 30 days, applying the indicator to the security's ask
price.
var history = IndicatorHistory(indicator, symbol, TimeSpan.FromDays(30), selector: Field.AskClose);
```

Some indicators require the prices of two securities to compute their value (for example, Beta). In this case, pass a list of the `Symbol` objects to the method.

```C#
public class CFDMultiAssetIndicatorHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 19);
        SetEndDate(2024, 12, 31);
        // Add the target and reference securities.
        var targetSymbol = AddCfd("XAUUSD").Symbol;
        var referenceSymbol = AddCfd("SPX500USD").Symbol;
        // Create a 21-period Beta indicator.
        var beta = new Beta("", targetSymbol, referenceSymbol, 21);
        // Get the historical values of the indicator over the last 10 trading days.
        var history = IndicatorHistory(beta, new[] {targetSymbol, referenceSymbol}, 10, Resolution.Daily);
        // Get the average Beta value.
        var avgBeta = history.Average(indicatorDataPoint => indicatorDataPoint.Current.Value);
    }
}
```

If you already have a list of Slice objects, you can pass them to the `IndicatorHistory` method to avoid the internal history request.

```C#
var slices = History(new[] {symbol}, 30, Resolution.Daily);
var history = IndicatorHistory(indicator, slices);
```

Examples

The following examples demonstrate some common practices for trading CFD with historical data.

Example 1: Demand Force Following

The following algorithm measures the historical bid-ask spread of XAUUSD CFD contract. If the current spread is higher than the historical spread, meaning the demand force is lower than usual, we sell the contract; else, we buy. To realize the directional speculation, we apply a 2% take profit risk management model and a 1% trailing stop loss risk management model.

C#

```
public class CFDHistoricalDataAlgorithm : QCAlgorithm
{
    private const decimal Threshold = 2m;
    private decimal _spreadMean = 0m;
    private decimal _spreadSd = 1m;
    private Symbol _xauusd;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);

        // Request the XAUUSD CFD contract for trade.
        _xauusd = AddCfd("XAUUSD").Symbol;

        // Update the spread mean and SD to calculate the Z-score accurately every day.
        Schedule.On(
            DateRules.EveryDay(_xauusd),
            TimeRules.At(0, 0),
            UpdateSpread
        );

        // Take profit at 2%.
        AddRiskManagement(new MaximumUnrealizedProfitPercentPerSecurity(0.02m));
        // Trailing stop loss at 1%.
        AddRiskManagement(new TrailingStopRiskManagementModel(0.01m));
    }

    private void UpdateSpread()
    {
        // Obtain historical quote data to obtain the mean and SD of the bid-ask spread.
        var history = History<QuoteBar>(_xauusd, 390, Resolution.Minute)      // minute bar of 1 day
            .Where(x => x.Ask != null && x.Bid != null)
            .ToList();
        if (history.Count > 0)
        {
            var spread = history.Select(x => x.Ask.Close - x.Bid.Close).ToList();
            _spreadMean = spread.Average();
            _spreadSd = Convert.ToDecimal(Math.Sqrt(spread.Sum(x => Math.Pow((double)(x - _spreadMean), 2)) /
spread.Count));
        }
    }

    public override void OnData(Slice slice)
    {
        // Obtain the current bid-ask spread as a trade signal.
        if (slice.QuoteBars.TryGetValue(_xauusd, out var quote))
        {
            if (quote.Ask != null && quote.Bid != null)
            {
                var spread = quote.Ask.Close - quote.Bid.Close;
                var z = (spread - _spreadMean) / _spreadSd;

                // If the spread is lower than expected, expect demand to be higher than usual, and buy XAUUSD.
                if (z <= -Threshold)
                {
                    SetHoldings(_xauusd, 1m);
                }
                // If the spread is higher than expected, expect demand to be lower than usual and sell XAUUSD.
                else if (z >= Threshold)
                {
                    SetHoldings(_xauusd, -1m);
                }
            }
        }
    }
}
```

Historical Data

Alternative Data

Introduction

Alternative datasets provide signals to inform trading decisions. To view all the alternative datasets available on QuantConnect, see the Dataset Market . This page explains how to get historical data for alternative datasets.

Data Points

To get a list of historical alternative data, call the `History< alternativeDataClass >` method with the dataset `Symbol` .

```
C#
public class AlternativeDataHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 20);
        SetEndDate(2024, 12, 31);
        // Get the Symbol of a dataset.
        var datasetSymbol = AddData<Fred>("RVXCLS").Symbol;
        // Get the trailing 5 days of Fred data.
        var history = History<Fred>(datasetSymbol, 5, Resolution.Daily);
        // Iterate through the historical data points.
        foreach (var dataPoint in history)
        {
            var t = dataPoint.EndTime;
            var value = dataPoint.Value;
        }
    }
}
```

Some alternative datasets provide multiple entries per asset per time step . For example, the US Regulatory Alerts dataset can provide multiple alerts per day. In this case, use a nested loop to access all the data point attributes.

```
C#
public class RegalyticsHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 20);
        SetEndDate(2024, 12, 31);
        // Get the all the Regalytics articles that were published over the last day.
        var datasetSymbol = AddData<RegalyticsRegulatoryArticles>("REG").Symbol;
        var history = History<RegalyticsRegulatoryArticles>(datasetSymbol, 1, Resolution.Daily);
        // Iterate through each day of articles.
        foreach (var articles in history)
        {
            var t = articles.EndTime;
            // Get the unique alert types for this day.
            var altertTypes = articles.Select(article => (article as
RegalyticsRegulatoryArticle).AlertType).Distinct().ToList();
        }
    }
}
```

Universes

To get historical data for an alternative data universe , call the `History` method with the `Universe` object.

```C#
public class AltDataUniverseHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 23);
        SetEndDate(2024, 12, 31);
        // Add a universe of US Equities based on an alternative dataset.
        var universe = AddUniverse<BrainStockRankingUniverse>();
        // Get 5 days of history for the universe.
        var history = History(universe, TimeSpan.FromDays(5));
        // Iterate through each day of the universe history.
        foreach (var altCoarse in history)
        {
            // Iterate through each asset in the universe on this day and access its data point attributes.
            foreach (BrainStockRankingUniverse stockRanking in altCoarse)
            {
                var symbol = stockRanking.Symbol;
                var t = stockRanking.EndTime;
                var rank2Days = stockRanking.Rank2Days;
            }
        }
    }
}
```

Slices

To request `Slice` objects of historical data, call the `History` method. If you pass a list of `Symbol` objects, it returns data for all the alternative datasets that the `Symbol` objects reference.

```C#
public class SliceHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 23);
        SetEndDate(2024, 12, 31);
        // Add an alternative dataset.
        var symbol = AddCrypto("BTCUSD", Resolution.Daily, Market.Bitfinex).Symbol;
        var datasetSymbol = AddData<BitcoinMetadata>(symbol).Symbol;
        // Get the latest 3 data points of some alternative dataset(s), packaged into Slice objects.
        var history = History(new[] { datasetSymbol }, 3);
        // Iterate through each Slice and get the alternative data points.
        foreach (var slice in history)
        {
            var t = slice.Time;
            var hashRate = slice[datasetSymbol].HashRate;
        }
    }
}
```

If you don't pass any `Symbol` objects, it returns data for all the data subscriptions in your algorithm, so the result may include more than just alternative data.

```
// Get the latest 3 data points of all the securities/datasets in the algorithm, packaged into Slice objects.
var history = History(3);
// Iterate through each Slice and get the synchronized data points at each moment in time.
foreach (var slice in history)
{
    var t = slice.Time;
    if (slice.ContainsKey(symbol))
    {
        var price = slice[symbol].Price;
    }
    if (slice.ContainsKey(datasetSymbol))
    {
        var hashRate = ((BitcoinMetadata)slice[datasetSymbol]).HashRate;
    }
}
```

C#

When your history request returns `Slice` objects, the `Time` properties of these objects are based on the algorithm time zone, but the `EndTime` properties of the individual data objects are based on the data time zone . The `EndTime` is the end of the sampling period and when the data is actually available.

Sparse Datasets

A sparse dataset is a dataset that doesn't have data for every time step of its resolution. For example, the US Energy Information Administration (EIA) datasets have a daily resolution but the data for the "U.S. Ending Stocks of Finished Motor Gasoline in Thousand Barrels (Mbbl)" series only updates once a week. So when you request the trailing 30 days of historical data for it, you only get a few data points.

```
public class SparseDatasetHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 23);
        SetEndDate(2024, 12, 31);
        // Add a sparse dataset. In this case, the default resolution is daily.
        var symbol = AddData<USEnergy>("PET.WGFSTUS1.W").Symbol;
        // Get 30 days of history for the dataset.
        var history = History(symbol, 30);
        // Iterate through each data point.
        foreach (var dataPoint in history)
        {
            var t = dataPoint.EndTime;
            var weeklyImports = dataPoint.Value;
        }
    }
}
```

C#

Most alternative datasets have only one resolution, which is usually daily. To check if a dataset is sparse and to view its resolution(s), see the documentation in the Dataset Market .

Examples

The following examples demonstrate some common practices for trade with historical Alternative Data.

Example 1: Inverted Yield Curve

The following algorithm trades the smoothened yield curve difference, calculated from the difference between 10-year yield and 2-year yield historical data and smoothened by the exponential moving average. If the yield curve is inverted, short SPY from this market downturn indicator.

C#

```
public class AlternativeHistoricalDataAlgorithm : QCAlgorithm
{
    private const decimal SmoothingFactor = 2m / (1m + 250m);
    private Symbol _spy, _yieldCurveSymbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);

        // Requesting yield curve data for trade signal generation (inversion)
        _yieldCurveSymbol = AddData<USTreasuryYieldCurveRate>("USTYCR").Symbol;
        // Request the XAUUSD CFD contract for trade.
        _spy = AddEquity("SPY").Symbol;

        // Ignore orders with very small sizes.
        Settings.MinimumOrderMarginPortfolioPercentage = 0.01m;

        // Trade every day after the market opens with an updated smoothened yield curve difference.
        Schedule.On(
            DateRules.EveryDay(_spy),
            TimeRules.AfterMarketOpen(_spy, 1),
            Trade
        );
    }

    private void Trade()
    {
        // Obtain historical quote data to obtain the mean and SD of the bid-ask spread.
        var history = History<USTreasuryYieldCurveRate>(_yieldCurveSymbol, 250, Resolution.Daily)
            .Where(x => x.TenYear.HasValue && x.TwoYear.HasValue)
            .ToList();
        if (history.Count > 0)
        {
            var diffs = history.Select(x => x.TenYear.Value - x.TwoYear.Value).ToList();
            var smoothedDiff = diffs.Aggregate((ema, nextQuote) => SmoothingFactor * nextQuote + (1m -
SmoothingFactor) * ema);

            // If the smoothened yield curve difference is inverted, short SPY with the downside signal.
            SetHoldings(_spy, smoothedDiff < 0 ? -1m : 1m);
        }
    }
}
```

Historical Data

Custom Data

Introduction

You can import external datasets into your algorithm to use alongside other datasets from the Dataset Market. This page explains how to get historical data for custom datasets. Before you can get historical data for the dataset, define the `GetSource` and `Reader` methods of the custom data class. For examples of custom dataset implementations, see Key Concepts .

Slices

To request `Slice` objects of historical data, call the `History` method. If you pass a list of `Symbol` objects, it returns data for all the custom datasets that the `Symbol` objects reference.

```
C#
// Get the latest 3 data points of some custom datasets, packaged into Slice objects.
var history = History(datasetSymbols, 3);
```

If you don't pass any `Symbol` objects, it returns data for all the data subscriptions in your notebook, so the result may include more than just custom data.

```
C#
// Get the latest 3 data points of all the securities/datasets in the notebook, packaged into Slice objects.
var history = History(3);
```

When your history request returns `Slice` objects, the `Time` properties of these objects are based on the algorithm time zone, but the `EndTime` properties of the individual data objects are based on the data time zone . The `EndTime` is the end of the sampling period and when the data is actually available.

Data Points

To get a list of historical data points for a custom dataset, call the `History< customDatasetClass >` method with the dataset `Symbol` . For an example definition of a custom data class, see the CSV Format Example .

```
C#
public class CustomSecurityHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2014, 7, 10);
        // Add a custom dataset and save a reference to it's Symbol.
        var datasetSymbol = AddData<MyCustomDataType>("MyCustomDataType", Resolution.Daily).Symbol;
        // Get the trailing 5 days of MyCustomDataType data.
        var history = History<MyCustomDataType>(datasetSymbol, 5, Resolution.Daily);
    }
}
```

Universes

To get historical data for a custom data universe , call the `History` method with the `Universe` object. For an example definition of a custom data universe class, see the CSV Format Example .

```C#
public class CustomDataUniverseHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2017, 7, 9);
        // Add a universe from a custom data source and save a reference to it.
        var universe = AddUniverse<StockDataSource>(
            "myStockDataSource", Resolution.Daily, data => data.Select(x => x.Symbol)
        );
        // Get the historical universe data over the last 5 days.
        var history = History(universe, TimeSpan.FromDays(5)).Cast<StockDataSource>().ToList();
        // Iterate through each day in the universe history and count the number of constituents.
        foreach (var stockData in history)
        {
            var t = stockData.Time;
            var size = stockData.Symbols.Count;
        }
    }
}
```

Missing Data Points

History requests for a trailing number of data samples return data based on the market hours of assets. The default market hours for custom securities is to be always open. Therefore, history requests for a trailing number of data samples may return fewer samples than you expect. To set the market hours of the dataset, see Market Hours .

Examples

The following examples demonstrate some common practices for trading with historical custom data.

Example 1: Custom Universe Data

The following algorithm trades based on a custom universe dataset. It obtains historical data on the universe's weights and uses the exponential moving average as the investment weight.

C#

```
public class CustomUniverseExampleAlgorithm : QCAlgorithm
{
    private const decimal SmoothingFactor = 2m / (1m + 5m);
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2016, 1, 1);
        SetEndDate(2025, 1, 1);
        // Add a universe that reads from the Object Store.
        _universe = AddUniverse<CustomUniverseData>("CustomUniverse", Resolution.Daily, (altCoarse) => {
            return altCoarse.Select(d => d.Symbol);
        });
        // Add a Scheduled Event to rebalance the portfolio.
        var spy = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
        Schedule.On(
            DateRules.EveryDay(spy),
            TimeRules.AfterMarketOpen(spy, 1),
            Rebalance
        );
    }

    private void Rebalance()
    {
        // We invest by the EMA of the given weight.
        var history = History(_universe, 5, Resolution.Daily);
        var symbols = history.Select(x => x.Symbol).ToHashSet();
        var weights = symbols.Select(symbol => {
                var ema = history.Where(x => x.Symbol == symbol)
                    .Select(x => (x as CustomUniverseData).Weight)
                    .Aggregate((ema, nextQuote) => SmoothingFactor * nextQuote + (1m - SmoothingFactor) * ema);
                return (symbol, ema);
            }).ToList();
        // Normalized weights.
        var weightSum = weights.Sum(x => x.Item2);
        var targets = weights.Select(x => new PortfolioTarget(x.Item1, x.Item2 / weightSum)).ToList();
        SetHoldings(targets, liquidateExistingHoldings: true);
    }
}

public class CustomUniverseData : BaseDataCollection
{
    public decimal Weight;
    public override DateTime EndTime
    {
        // Set the data period to 1 day.
        get { return Time + QuantConnect.Time.OneDay; }
        set { Time = value - QuantConnect.Time.OneDay; }
    }

    public override SubscriptionDataSource GetSource(SubscriptionDataConfig config, DateTime date, bool isLiveMode)
    {
        // Define the location and format of the data file.
        return new SubscriptionDataSource(
            "portfolioTargets.csv", SubscriptionTransportMedium.ObjectStore, FileFormat.Csv
        );
    }

    public override BaseData Reader(SubscriptionDataConfig config, string line, DateTime date, bool isLiveMode)
    {
        // Skip the header row.
        if (!Char.IsDigit(line[0]))
        {
            return null;
        }
        // Split the line by each comma.
        var items = line.Split(",");

        // Parse the data from the CSV file.
        return new CustomUniverseData
        {
            EndTime = Parse.DateTimeExact(items[0], "yyyy-MM-dd"),
            Symbol = Symbol.Create(items[1], SecurityType.Equity, Market.USA),
            Weight = decimal.Parse(items[2], NumberStyles.Any, CultureInfo.InvariantCulture)
        };
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm
HistoryAlgorithm.cs C#

Historical Data

Universe Data

Introduction

Universe selection is the process of selecting a basket of assets you may trade. Dynamic universe selection increase diversification and decrease selection bias in your algorithm. To get the history of a universe, call the `History` method with the universe object. Note that the `History` method doesn't filter the universe with your filter function. It simply returns all of the assets and data points in the universe dataset.

US Equity Fundamentals

The following example gets the history of a US Equity fundamental universe :

C#

```
public class USEquityFundamentalUniverseHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 23);
        SetEndDate(2024, 12, 31);
        // Add a universe of US Equities based on the Fundamentals dataset.
        var universe = AddUniverse(fundamentals => fundamentals.Select(f => f.Symbol));
        // Get 5 days of history for the universe.
        var history = History(universe, TimeSpan.FromDays(5));
        // Iterate through each day of the universe history.
        foreach (var fundamentals in history)
        {
            // Iterate through each asset in the universe on this day and access its data point attributes.
            foreach (Fundamental f in fundamentals)
            {
                var symbol = f.Symbol;
                var t = f.EndTime;
                var peRatio = f.ValuationRatios.PERatio;
            }
        }
    }
}
```

US Equity ETF Constituents

The following example gets the history of a US Equity ETF constituents universe :

C#

```
public class ETFConstituentUniverseHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 23);
        SetEndDate(2024, 12, 31);
        // Add a universe of US Equities based on the constituents of an ETF.
        var universe = AddUniverse(Universe.ETF("SPY"));
        // Get 5 days of history for the universe.
        var history = History(universe, TimeSpan.FromDays(5));
        // Iterate through each day of the universe history.
        foreach (var constituents in history)
        {
            // Select the 2 assets with the smallest weights in the ETF on this day.
            var dailyLargest = constituents.Select(c => c as ETFConstituentData).OrderBy(c => c.Weight).Take(2);
        }
    }
}
```

US Equity Alternative Data

The following example gets the history of a US Equity alternative data universe :

```
C#
public class USEquityAltDataUniverseHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 23);
        SetEndDate(2024, 12, 31);
        // Add a universe of US Equities based on an alternative dataset.
        var universe = AddUniverse<BrainStockRankingUniverse>();
        // Get 5 days of history for the universe.
        var history = History(universe, TimeSpan.FromDays(5));
        // Iterate through each day of the universe history.
        foreach (var altCoarse in history)
        {
            // Select the asset with the greatest value each day.
            var dailyWinner = altCoarse
                .Select(c => c as BrainStockRankingUniverse)
                .OrderByDescending(c => c.Value)
                .First();
        }
    }
}
```

Equity Options

The following example gets the history of an Equity Options universe :

```
C#
public class EquityOptionUniverseHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 23);
        SetEndDate(2024, 12, 31);
        // Add a universe of Equity Option contracts.
        var option = AddOption("SPY");
        // Get 5 days of history for the universe.
        var history = History<OptionUniverse>(option.Symbol, TimeSpan.FromDays(5));
        // Iterate through each day of the universe history.
        foreach (var chain in history)
        {
            var t = chain.EndTime;
            // Select contracts that have open interest > 0 on this day.
            var openContracts = chain.Data
                .Select(contract => contract as OptionUniverse)
                .Where(contract => contract.OpenInterest > 0m);
        }
    }
}
```

Index Options

The following example gets the history of an Index Options universe :

```
C#
public class IndexOptionsUniverseHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 23);
        SetEndDate(2024, 12, 31);
        // Add a universe of Index Option contracts.
        var option = AddIndexOption("VIX");
        // Get 5 days of history for the universe.
        var history = History<OptionUniverse>(option.Symbol, TimeSpan.FromDays(5));
        // Iterate through each day of the universe history.
        foreach (var chain in history)
        {
            var t = chain.EndTime;
            // Calculate the trading volume across all the contracts on this day.
            var dailyVolume = chain.Data
                .Select(contract => contract as OptionUniverse)
                .Sum(contract => contract.Volume);
        }
    }
}
```

Crypto

The following example gets the history of a Crypto universe :

```
C#
public class CryptoUniverseHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 23);
        SetEndDate(2024, 12, 31);
        // Add a universe of Cryptocurrencies.
        var universe = AddUniverse(CryptoUniverse.Coinbase(data => data.Select(x => x.Symbol)));
        // Get 5 days of history for the universe.
        var history = History(universe, TimeSpan.FromDays(5));
        // Iterate through each day of the universe history.
        foreach (var universeDay in history)
        {
            var t = universeDay.EndTime;
            // Select the Crypto with the largest daily return on this day.
            var topGainer = universeDay
                .Select(c => c as CryptoUniverse)
                .OrderByDescending(c => (c.Close - c.Open) / c.Open)
                .First();
        }
    }
}
```

Crypto Market Cap

The following example gets the history of a Crypto market cap universe :

```C#
public class CryptoMarketCapUniverseHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 24);
        SetEndDate(2024, 12, 31);
        // Add a universe of Cryptocurrencies.
        var universe = AddUniverse<CoinGecko>();
        // Get 365 days of history for the universe.
        var history = History(universe, TimeSpan.FromDays(365));
        // Iterate through each day of the universe history.
        foreach (var universeDay in history)
        {
            var t = universeDay.EndTime;
            // Select the Crypto with the largest market cap on this day.
            var largestCoin = universeDay.Select(c => c as CoinGecko).OrderByDescending(c => c.MarketCap).First();
        }
    }
}
```

Custom Data

The following example gets the history of a custom data universe :

C#

```
// Define the custom data class outside of the algorithm class.
public class StockDataSource : BaseData
{
    public List<string> Symbols { get; set; } = new();
    public override DateTime EndTime => Time.AddDays(1);

    public override SubscriptionDataSource GetSource(SubscriptionDataConfig config, DateTime date, bool isLiveMode)
    {
        return new SubscriptionDataSource(
            "https://raw.githubusercontent.com/QuantConnect/Documentation/master/Resources/datasets/custom-data/csv-
universe-example.csv",
            SubscriptionTransportMedium.RemoteFile,
            FileFormat.Csv
        );
    }

    public override BaseData Reader(SubscriptionDataConfig config, string line, DateTime date, bool isLiveMode)
    {
        var stocks = new StockDataSource();
        stocks.Symbol = config.Symbol;
        try
        {
            var csv = line.Split(',');
            stocks.Time = DateTime.ParseExact(csv[0], "yyyyMMdd", null);
            stocks.Symbols.AddRange(csv.Skip(1));
        }
        catch { return null; }
        return stocks;
    }
}

public class CustomDataUniverseHistoryAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add a custom data universe.
        var universe = AddUniverse<StockDataSource>("StockDataSource", Resolution.Daily);
        // Get the universe history.
        var history = History(universe, new DateTime(2018, 1, 1), new DateTime(2018, 1, 10)).Cast<StockDataSource>();
        // Iterate through each day in the universe history and count the number of constituents.
        foreach (var stockData in history)
        {
            var t = stockData.Time;
            var size = stockData.Symbols.Count;
        }
    }
}
```

Examples

The following examples demonstrate some common practices for trading with historical universe data.

Example 1: Historical Ranking Prediction

The following example forms an equal-weighted portfolio at the start of each month with the 5 assets that have the greatest 21-day ranking in the Brain ML Stock Ranking dataset.

C#

```
public class UniverseHistoricalDataAlgorithm : QCAlgorithm
{
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);
        Settings.SeedInitialPrices = true;

        // Add a monthly update to the universe of BrainStockRanking data.
        UniverseSettings.Schedule.On(DateRules.MonthStart("SPY"));
        _universe = AddUniverse<BrainStockRankingUniverse>();

        // Trade every day after the market opens with an updated smoothened yield curve difference.
        Schedule.On(
            DateRules.MonthStart("SPY"),
            TimeRules.At(9, 31),
            Trade
        );

        SetWarmUp(1);
    }

    private void Trade()
    {
        // Get the last 5 bars of history for the universe.
        var symbolRanks = new Dictionary<Symbol, decimal>();
        var history = History(_universe, 5, Resolution.Daily, fillForward: false);
        foreach (var bar in history)
        {
            foreach (var data in bar.Select(c => c as BrainStockRankingUniverse)
                .Where(x => x.Rank21Days.HasValue))
            {
                if (!symbolRanks.TryGetValue(data.Symbol, out var rank))
                {
                    symbolRanks[data.Symbol] = 0m;
                }
                symbolRanks[data.Symbol] += data.Rank21Days.Value;
            }
        }
        // Obtain the 5 highest rank stocks to invest on equally.
        var targets = symbolRanks.OrderByDescending(x => x.Value)
            .Take(5)
            .Select(x => new PortfolioTarget(x.Key, 0.2m))
            .ToList();
        SetHoldings(targets, liquidateExistingHoldings: true);
    }
}
```

Historical Data

Warm Up Periods

Introduction

LEAN supports an automated fast-forward system called "Warm Up". It simulates winding back the date you deployed your algorithm by a specific time period. In backtests, it changes the start date of your algorithm to an earlier time. In live trading, it fetches data from before your deployment and fast forwards through your strategy up to the present day. Warm Up is a great way to prepare your algorithm and its indicators for trading. You can't place trades during the warm-up period because the data feed is replaying historical data for setting algorithm state.

Set Warm Up Periods

You can set a warm-up period based on a number of bars or a period of time.

Trailing Data Samples

To set a warm-up based on a trailing number of data samples, in your algorithm's `Initialize` method, call the `SetWarmUp` method with an integer argument.

```C#
// Feed in 100 bars before start date
SetWarmUp(100);
```

LEAN calculates the start of the warm-up period for each of your security subscriptions by using the number of bars you specify, the resolution of each security, and the trading calendar of each security. After it calculates the start time of the warm-up period for each security, it sets the earliest start time as the start of the algorithm warm-up period. For instance, in the following example, the warm-up period consists of 100 minute resolution bars for SPY and as many second resolution bars for BTCUSD that occur from the start of the SPY warm-up period to the algorithm `StartDate` .

```C#
public class WarmUpWithTrailingDataSamplesAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        AddEquity("SPY", Resolution.Minute, fillForward: false);
        AddCrypto("BTCUSD", Resolution.Second, fillForward: false);
        SetWarmUp(100);
    }
}
```

To use a specific resolution of data for the warm-up period, pass a `resolution` argument to the `SetWarmUp` method.

```C#
// Feed in data for 100 trading days of daily data before the start date
SetWarmUp(100, Resolution.Daily);
```

If you pass an integer and a resolution, the warm-up period consists of the number of bars and resolution you set, regardless of the resolution of your data subscriptions. In this case, LEAN calculates the start of the warm-up period for each of your security

subscriptions just like it does when you only pass an integer argument. After it calculates the start time of the warm-up period for each security, it sets the earliest start time as the start of the algorithm warm-up period.

If you pass a resolution argument and you registered indicators or consolidators for automatic updates, the warm-up period resolution must be less than or equal to the lowest resolution of your automatic indicators and consolidators. For instance, in the following example, the indicators use minute resolution data, so you can set the warm-up period to use tick, second, or minute resolution data.

```C#
public class WarmUpWithTrailingDataSamplesAndResolutionAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        var spy = AddEquity("SPY", Resolution.Minute, fillForward: false).Symbol;
        var btc = AddCrypto("BTCUSD", Resolution.Second, fillForward: false).Symbol;

        var spySma = SMA(spy, 10);
        var btcSMA = SMA(btc, 10, Resolution.Minute);

        SetWarmUp(100, Resolution.Minute);
    }
}
```

Trailing Time Periods

To set a warm-up based on a trailing time period, in your algorithm's `Initialize` method, call the `SetWarmUp` method with a `TimeSpan` . The warm-up period consists of that `TimeSpan` before the start date and pipes in data that matches the resolution of your data subscriptions.

```C#
// Wind time back 7 days from start
SetWarmUp(TimeSpan.FromDays(7));
```

To use a specific resolution of data for the warm-up period, pass a `resolution` argument to the `SetWarmUp` method.

```C#
// Feed in 100 days of daily data before the start date
SetWarmUp(TimeSpan.FromDays(100), Resolution.Daily);
```

If you pass a `TimeSpan` and a resolution, the warm-up period consists of the time period and resolution you set, regardless of the resolution of your data subscriptions.

Warm Up Status

You can't place trades during the warm-up period because the data feed is replaying historical data for setting algorithm state. To check if the algorithm is currently in a warm up period, use the `IsWarmingUp` property.

```C#
public class WarmUpStatusAlgorithm : QCAlgorithm
{
    private ExponentialMovingAverage _ema;
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        var symbol = AddEquity("IBM", Resolution.Daily).Symbol;
        _ema = EMA(symbol, 100);
        SetWarmup(100);
    }

    public override void OnData(Slice data)
    {
        if (IsWarmingUp)
        {
            return;
        }
        Plot("EMA", "Is ready", _ema.IsReady ? 1 : 0);
    }
}
```

Event Handler

The `OnWarmupFinished` event handler executes when the warm up period ends, even if you don't set a warm up period.

```C#
public override void OnWarmupFinished()
{
    // Done warming up
}
```

Missing Data Points

There are several cases where you may get fewer data points from during warm-up than you expect.

Illiquid Securities

The default behavior for LEAN is to fill-forward the data. However, if an asset had no trading activity during the time period you request historical data for, you won't get any data for that asset.

Requesting Data Before an Asset's IPO

All assets have a day when they first started trading. If your warm-up period includes time before an asset's initial public offering (IPO) date, you won't get any data before the IPO date because the data doesn't exist.

Requesting Data Outside Market Hours

Warming up with a trailing number of data samples means the data is based on the market hours of assets, so they can wrap across multiple trading days to gather the amount of data you request. In contrast, warming up with a trailing time period provides means the data is based on a rolling calendar time. In this latter types of warm-up, if the period of time is outside of the market hours of an asset, you won't get any data.

Data Issues

Data issues are generally a result of human error or from mistakes in the data collection process. Any QuantConnect member can report a data issue , which our Data Team works to quickly resolve. However, while the data is still missing, your warm-up won't be able to provide the data. To view the list of current data issues, log in to the Algorithm Lab and then, in the left navigation bar, click Datasets > Data Issues .

Data Provider Limitations

When you run warm-up, consider the functionality of the data provider you use. For example, the QuantConnect data provider has quotas on the amount of data you can request and the Interactive Brokers data provider can be slow, so it can timeout if you run warm-up with too many assets. For more information about all the data providers, see their respective documentation in Cloud Platform , Local Platform , or the CLI .

Warm Up vs History Requests

Warm-up and history requests both provide historical data. The approach you take depends on your specific use case. A benefit to using warm-up is that it only takes one line of code to pump historical data into your algorithm, which processes universe selection and updates all the indicators. A consequence to using warm-up is that it may provide more historica data than you actually need since it provides data on all the subscriptions in your algorithm. If you have derivative assets in your algorithm, warm-up can be very slow to process.

Examples

The following examples demonstrate some common practices for warm up.

Example 1: Warm Up Indicator

The following algorithm trades mean-reversion according to signals from a Bollinger Bands indicator. It uses warm up so the indicator is ready by the start date of the algorithm.

C#

```
public class WarmUpPeriodsAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private BollingerBands _bbands = new(20, 2);

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request SPY data for signal generation and trading.
        _spy = AddEquity("SPY", Resolution.Minute).Symbol;

        // The candlestick patterns are based on a daily consolidated trade bar.
        var consolidator = new TradeBarConsolidator(TimeSpan.FromDays(1));
        // Subscribe to update the indicators with the 1-day consolidator automatically.
        RegisterIndicator(_spy, _bbands, consolidator);
        // Add an event handler on candlestick indicators that are updated to trade the indicator.
        _bbands.Updated += OnUpdated;

        // Warm up the indicator and the SPY price data.
        SetWarmUp(20, Resolution.Daily);
    }

    private void OnUpdated(object sender, IndicatorDataPoint point)
    {
        var indicator = sender as BollingerBands;
        if (!indicator.IsReady) return;

        var holdings = Portfolio[_spy];

        // Trade mean-reversal of the Bollinger Band.
        if (holdings.Price > indicator.UpperBand && !holdings.IsShort)
        {
            SetHoldings(_spy, -0.5m);
        }
        else if (holdings.Price < indicator.LowerBand && !holdings.IsLong)
        {
            SetHoldings(_spy, 0.5m);
        }
    }
}
```

Example 2: Warm Up Universe Selection

The following algorithm forms an equal-weighted portfolio with the 100 most liquid US Equities that are above their 60-day Exponential Moving Average (EMA). It uses the warm-up feature to prime the indicators in advance so it can start trading by the start date of the algorithm.

C#

```
public class WarmUpPeriodsAlgorithm : QCAlgorithm
{
    private Universe _universe;
    private Dictionary<Symbol, ExponentialMovingAverage> _emaBySymbol = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        Settings.SeedInitialPrices = true;

        // Universe filtered by only the ones with prices above EMA, suggesting an uptrend.
        _universe = AddUniverse(SelectionByEma);

        // Rebalance weekly.
        Schedule.On(
            DateRules.WeekStart(),
            TimeRules.At(9, 31),
            Rebalance
        );

        SetWarmUp(60, Resolution.Daily);
    }

    private IEnumerable<Symbol> SelectionByEma(IEnumerable<Fundamental> fundamentals)
    {
        foreach (var f in fundamentals)
        {
            // Create an EMA indicator for the symbol if it is unavailable.
            if (!_emaBySymbol.TryGetValue(f.Symbol, out var ema))
            {
                _emaBySymbol[f.Symbol] = ema = new ExponentialMovingAverage(60);
            }
            // Update the indicator by the updated price data to select on the updated data.
            ema.Update(f.EndTime, f.AdjustedPrice);
        }

        // We don't add new securities during the warm-up process
        if (IsWarmingUp) return Universe.Unchanged;

        return fundamentals
            // Only trades the top 100 liquid stocks since their trend capitalizes quicker.
            .OrderByDescending(f => f.DollarVolume)
            // Select the ones with the current price above EMA to filter for the uptrend stocks.
            .Where(f => _emaBySymbol[f.Symbol].IsReady && _emaBySymbol[f.Symbol] > f.AdjustedPrice)
            .Take(100)
            .Select(f => f.Symbol);
    }

    private void Rebalance()
    {
        // Equally invest in uptrend stocks to dissipate capital risk evenly.
        var weight = 1m / _universe.Selected?.Count ?? 0m;
        if (weight == 0) return;
        var targets = _universe.Selected.Select(symbol => new PortfolioTarget(symbol, weight)).ToList();
        SetHoldings(targets);
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        // Exit position if the equity is below EMA since it becomes a downgoing trend.
        changes.RemovedSecurities.DoForEach(r => Liquidate(r.Symbol));
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
WarmupAlgorithm.cs C#

Historical Data

Rolling Window

Introduction

A `RollingWindow` is an array of a fixed-size that holds trailing data. It's more efficient to use `RollingWindow` objects to hold periods of data than to make multiple historical data requests . With a `RollingWindow` , you just update the latest data point while a `History` call fetches all of the data over the period you request. `RollingWindow` objects operate on a first-in, first-out process to allow for reverse list access semantics with positive and negative indexing. Index 0 refers to the most recent item in the window and the largest index refers to the last item in the window. Index -1 refers to the latest item as well.

Create Rolling Windows

To create rolling windows, instantiate the `RollingWindow` object of a given type with a maximum size.

```
C#

closePriceWindow = new RollingWindow<decimal>(4);
tradeBarWindow = new RollingWindow<TradeBar>(2);
quoteBarWindow = new RollingWindow<QuoteBar>(2);
```

Add Data

To add data to a `RollingWindow` , call the `Add` method.

```
C#

closePriceWindow.Add(data["SPY"].Close);
tradeBarWindow.Add(data["SPY"]);
quoteBarWindow.Add(data["EURUSD"]);
```

To update the data at a specific index, set the value for that index. If the index doesn't currently exist, it increases the size and fills the empty indices with a default value (zero or `null`).

```
C#

closePriceWindow[0] = data["SPY"].Close;
tradeBarWindow[0] = data["SPY"];
quoteBarWindow[0] = data["EURUSD"];
```

Warm Up

To warm up a `RollingWindow` , make a history request and then iterate through the result to add the data to the `RollingWindow` .

```C#
public class WarmUpRollingWindowAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        var spy = AddEquity("SPY", Resolution.Daily).Symbol;
        var historyTradeBar = History<TradeBar>(spy, 10, Resolution.Daily);
        var historyQuoteBar = History<QuoteBar>(spy, 10, Resolution.Minute);

        // Warm up the close price and trade bar rolling windows with the previous 10-day trade bar data
        var closePriceWindow = new RollingWindow<decimal>(10);
        var tradeBarWindow = new RollingWindow<TradeBar>(10);
        foreach (var tradeBar in historyTradeBar)
        {
            closePriceWindow.Add(tradeBar.Close);
            tradeBarWindow.Add(tradeBar);
        }

        // Warm up the quote bar rolling window with the previous 10-minute quote bar data
        var quoteBarWindow = new RollingWindow<QuoteBar>(10);
        foreach (var quoteBar in historyQuoteBar)
        {
            quoteBarWindow.Add(quoteBar);
        }
    }
}
```

Check Readiness

To check if a `RollingWindow` is full, use its `IsReady` flag.

```C#
if (!closePriceWindow.IsReady)
{
    return;
}
```

Adjust Size

To adjust the `RollingWindow` size, set the `Size` property.

```C#
closePriceWindow.Size = 3;
tradeBarWindow.Size = 3;
quoteBarWindow.Size = 3;
```

When you decrease the size, it removes the oldest values that no longer fit in the `RollingWindow` . When you explicitly increase the `Size` member, it doesn't automatically add any new elements to the `RollingWindow` . However, if you set the value of an index in the `RollingWindow` and the index doesn't currently exist, it fills the empty indices with a default value (zero or `null`). For example, the following code increases the `Size` to 10, sets the 10th element to 3, and sets the 4th-9th elements to the default value:

```C#
closePriceWindow[9] = 3;
```

Access Data

`RollingWindow` objects operate on a first-in, first-out process to allow for reverse list access semantics with positive and negative indexing. Index 0 refers to the most recent item in the window and the largest index refers to the last item in the window. Index -1 refers to the latest item as well.

```
var currentClose = closePriceWindow[0];
var previousClose = closePriceWindow[1];
var oldestClose = closePriceWindow[closePriceWindow.Count-1];
oldestClose = closePriceWindow[-1];   // Negative indexing
```

To get the item that was most recently removed from the `RollingWindow` , use the `MostRecentlyRemoved` property.

```
var removedClose = closePriceWindow.MostRecentlyRemoved;
```

Combine with Indicators

The `Window` property of the indicators is a built-in `RollingWindow` that stores historical values. It holds a collection of `IndicatorDataPoint` objects, enabling quick access to the most recent historical indicator values for analysis, calculations, or comparisons in trading and financial strategies. Its default size 2, but you can adjust its size. To warm up the indicator and its `RollingWindow` , call the `IndicatorHistory` method with the window size as an argument.

```
public class IndicatorRollingWindowAlgorithm : QCAlgorithm
{
    // Manual SMA indicator example.
    private SimpleMovingAverage _sma = new(20);
    private Beta _beta;

    public override void Initialize()
    {
        var spy = AddEquity("SPY").Symbol;
        var qqq = AddEquity("QQQ").Symbol;

        // Automatic Beta indicator example.
        _beta = B(spy, qqq, 20);

        // Adjust the window size to 5 to access the previous 5 indicator data points.
        _sma.Window.Size = 5;
        _beta.Window.Size = 5;

        // Warm up the indicator and window for immediate usage.
        IndicatorHistory(_sma, spy, _sma.Window.Size, Resolution.Daily);
        // For indicators using 2+ symbols, use an iterable symbol list.
        IndicatorHistory(_beta, new[] { spy, qqq },  _beta.Window.Size, Resolution.Daily);
    }
}
```

Combine with Consolidators

To store a history of consolidated bars , in the consolidation handler, add the consolidated bar to the `RollingWindow` .

```
_consolidator.DataConsolidated += (sender, consolidatedBar) => tradeBarWindow.Add(consolidatedBar);
```

Market Session

The `session` property of a `Security` object contains the open, high, low, close, volume, and open interest of trading sessions. The `Session` object is a `RollingWindow` object with size of zero by default. To enable market session tracking, set the size of the `Session` object to 2, so the previous session data is at index 1.

```
var session = Securities[_symbol].Session;
session.Size = 2;
var todayOpen = session.Open;
var todayHigh = session.High;
var todayLow = session.Low;
var todayCumulativeVolume = session.Volume;
if (session.IsReady)
{
    var yesterdayClose = session[1].Close;
}
```

Increase the size of the `Session` object to store more session data. You can use cached data in the `session` property instead of requesting historical daily data .

```
var equity = AddEquity("SPY");
// Increase the session lookback to 252 from default (0)
equity.Session.Size = 252;
// Get all closing prices in the session history
var closes = session.OrderBy(x => x.EndTime).Select(x => x.Close);
```

Cast to Other Types

You can cast a `RollingWindow` to a list. If you cast it to a list, reverse the list so the most recent element is at the last index of the list. This is the order the elements would be in if you added the elements to the list with the `Add` method.

```
var closes = closePriceWindow.Reverse().ToList();
```

Delete Data

To remove all of the elements from a `RollingWindow` , call the `Reset` method.

```
closePriceWindow.Reset();
```

Examples

The following examples demonstrate some common practices for rolling windows.

Example 1: Price Actions

The following algorithm saves the trailing 3 `TradeBar` objects into a `RollingWindow` . When it identifies a volume contraction breakout price action pattern on the SPY, it buys to ride on the capital inflow. To exit positions, it places a 2% take profit and 1% stop loss order in the `OnOrderEvent` method.

```
public class RollingWindowAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    // Set up a rolling window to hold the last 3 trade bars for price action detection as the trade signal.
    private RollingWindow<TradeBar> _windows = new(3);

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Add SPY data for signal generation and trading.
        _spy = AddEquity("SPY", Resolution.Minute).Symbol;
```

```
        // Warm up the rolling window.
        var history = History<TradeBar>(_spy, 3, Resolution.Minute);
        foreach (var bar in history)
        {
            _windows.Add(bar);
        }
    }

    public override void OnData(Slice slice)
    {
        if (slice.Bars.TryGetValue(_spy, out var bar))
        {
            // Trade the price action if the previous bars fulfill a contraction breakout.
            if (ContractionAction() && BreakoutAction(bar.Close))
            {
                SetHoldings(_spy, 0.5m);
            }

            // Add the current bar to the window.
            _windows.Add(bar);
        }
    }

    private bool ContractionAction()
    {
        // We trade contraction type price action, where the buying preesure is increasing.
        // 1. The last 3 bars are green.
        // 2. The price is increasing in trend.
        // 3. The trading Volume is increasing as well.
        // 4. The range of the bars are decreasing.
        return _windows[2].Close > _windows[2].Open &&
            _windows[1].Close > _windows[1].Open &&
            _windows[0].Close > _windows[0].Open &&
            _windows[0].Close > _windows[1].Close && _windows[1].Close > _windows[2].Close &&
            _windows[0].Volume > _windows[1].Volume && _windows[1].Volume > _windows[2].Volume &&
            _windows[2].Close - _windows[2].Open > _windows[1].Close - _windows[1].Open &&
            _windows[1].Close - _windows[1].Open > _windows[0].Close - _windows[0].Open;
    }

    private bool BreakoutAction(decimal currentPrice)
    {
        // Trade breakout from contraction: the breakout should be much greater than the contracted range of the last
bar.
        return currentPrice - _windows[0].Close > (_windows[0].Close - _windows[0].Open) * 2m;
    }

    public override void OnOrderEvent(OrderEvent orderEvent)
    {
        if (orderEvent.Status == OrderStatus.Filled)
        {
            if (orderEvent.Ticket.OrderType == OrderType.Market)
            {
                // Stop loss order at 1%.
                var stopPrice = orderEvent.FillQuantity > 0m ? orderEvent.FillPrice * 0.99m : orderEvent.FillPrice *
1.01m;
                StopMarketOrder(_spy, -Portfolio[_spy].Quantity, stopPrice);
                // Take profit order at 2%.
                var takeProfitPrice = orderEvent.FillQuantity > 0m ? orderEvent.FillPrice * 1.02m :
orderEvent.FillPrice * 0.98m;
                LimitOrder(_spy, -Portfolio[_spy].Quantity, takeProfitPrice);
            }
            else if (orderEvent.Ticket.OrderType == OrderType.StopMarket || orderEvent.Ticket.OrderType ==
OrderType.Limit)
            {
                // Cancel open orders if the stop loss or take profit order fills.
                Transactions.CancelOpenOrders();
            }
        }
    }
}
```

Example 2: Bid-Ask Spread

The following algorithm trades the microeconomy of SPY's supply-demand relationship. It buys when the current bid-ask spread is less than average spread over the last 20 QuoteBar objects, indicating demand is approaching supply. When the

spread is greater than the average, it shorts SPY. To save the last spread values, it uses a `RollingWindow` .

```C#
public class RollingWindowAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    // Set up a rolling window to hold the last 20 bar's bid-ask spread for trade signal generation.
    private RollingWindow<decimal> _windows = new(20);

    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);

        // Add SPY data for signal generation and trading.
        _spy = AddEquity("SPY", Resolution.Minute).Symbol;

        // Warm up for the rolling window with quote data.
        var history = History<QuoteBar>(_spy, 20, Resolution.Minute);
        foreach (var bar in history)
        {
            _windows.Add(bar.Ask.Close - bar.Bid.Close);
        }
    }

    public override void OnData(Slice slice)
    {
        if (slice.QuoteBars.TryGetValue(_spy, out var bar))
        {
            // Update the window with the current bid-ask spread.
            var spread = bar.Ask.Close - bar.Bid.Close;
            _windows.Add(spread);

            // Buy if the current spread is smaller than average, indicating demand is approaching supply. Buy force
will drive up price.
            if (spread < _windows.Average())
            {
                SetHoldings(_spy, -0.5m);
            }
            // Short if the current spread is larger than average, indicating supply is gradually overwhelming
demand.
            else if (spread > _windows.Average())
            {
                SetHoldings(_spy, 0.5m);
            }
        }
    }
}
```

Example 3: EMA Acceleration

The following algorithm trades SPY based on the direction and acceleration of its exponential moving average (EMA). It buys when the EMA is increasing and accelerating upward, indicating a strong uptrend. It short sells when the EMA is decreasing and accelerating downward, indicating a strong downtrend. To compare the current EMA value to its previous values, the algorithm utilizes the EMA's internal `RollingWindow` .

```C#
public class RollingWindowAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private ExponentialMovingAverage _ema;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Add SPY data for trading.
        _spy = AddEquity("SPY", Resolution.Minute).Symbol;

        // Set up an automatic EMA indicator for trade signal generation.
        _ema = EMA(_spy, 20, Resolution.Daily);
        // Extend the size of the EMA's internal rolling window so we can access its historical values.
        _ema.Window.Size = 3;
        // Warm-up the indicator and its internal rolling window
        IndicatorHistory(_ema, _spy, _ema.Window.Size);

        // Schedule an event to rebalance SPY position at daily market open.
        Schedule.On(DateRules.EveryDay(_spy), TimeRules.AfterMarketOpen(_spy, 0), Rebalance);
    }

    private void Rebalance()
    {
        // Buy if the current EMA is increasing with acceleration, indicating a strong uptrend.
        if (_ema.Window[1] < _ema && _ema.Window[0] - _ema.Window[1] > _ema.Window[1] - _ema.Window[2])
        {
            SetHoldings(_spy, 0.5m);
        }
        // Short if the current EMA is decreasing with acceleration, indicating a strong downtrend.
        else if (_ema.Window[1] > _ema && _ema.Window[0] - _ema.Window[1] < _ema.Window[1] - _ema.Window[2])
        {
            SetHoldings(_spy, -0.5m);
        }
        // Liquidate if no strong trend indicated.
        else if (Portfolio.Invested)
        {
            Liquidate(_spy);
        }
    }
}
```

Example 4: Market Session as History

Using `Session` instead of history requests or rolling window to cache the last 252 OHLCV bars, the following algorithm buys SPY accordingly when the market opens above the 252-day max of closing prices.

C#

```
public class MarketSessionAsDailyHistory : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        var equity = AddEquity("SPY");

        // Increase the session lookback to 252 from default (0)
        // since our strategy break out the 252-day max of closing prices
        equity.Session.Size = 252;
        // Warm up with daily data for speed
        SetWarmUp(equity.Session.Size, Resolution.Daily);

        Schedule.On(DateRules.EveryDay("SPY"), TimeRules.AfterMarketOpen("SPY", 1), () =>
        {
            // Break-out strategy. If the market opens above the 252-day max of closing prices,
            // allocate all in SPY. Liquidate if it opens below the mean.

            var session = Securities["SPY"].Session;
            // The session is ready when the number of samples is greater than the size (252)
            if (IsWarmingUp || !session.IsReady)
                return;

            var closes = session.OrderBy(x => x.EndTime).Select(x => x.Close);
            var open = session.Open;
            var max = closes.Max();
            var mean = closes.Average();

            // Invest all in SPY if the open break out the 252-day max and liquidate below the mean
            if (open > max)
                SetHoldings("SPY", 1, tag: $"_open={open:F2}, _max={max:F2}");
            if (open < mean)
                Liquidate(symbol: "SPY", tag:$"_open={open:F2}, _mean={mean:F2}");
        });
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm
RollingWindowAlgorithm.cs C# MultipleSymbolConsolidationAlgorithm.cs C#

Historical Data

Common Errors

Introduction

Errors can occur when you request historical data. This page explains some of the most common ones you may encounter.

Indicator Readiness

Indicators can't produce a valid value until they have enough historical data to be ready . If your indicators require more data than what is currently available (for example, data before an asset's IPO), the `IndicatorHistory` method will return fewer data points than you request and the warm-up techniques for manual and automatic indicators won't be able to make your indicators ready.

Trading During Warm Up

You can't place trades during the warm-up period because the data feed is replaying historical data for setting algorithm state. If you try to place a trade during warm-up, LEAN throws the following error:

Backtest Handled Error: This operation is not allowed in Initialize initialize or during warm up: OrderRequest.Submit. Please move this code to the OnWarmupFinished() on_warmup_finished() method.

Numerical Precision

Some factor files have INF split values, which indicate that the stock has so many splits that prices can't be calculated with correct numerical precision. To allow history requests with these symbols, we need to move the starting date forward when reading the data or use raw data normalization . If there are numerical precision errors in the factor files for a security in your history request, LEAN throws the following error:

"Warning: when performing history requests, the start date will be adjusted if there are numerical precision errors in the factor files."

Live Intraday Data Source

In live trading, if you make a history request for minute data at noon and the history period covers the start of the previous day to the present moment, the data from the previous day will be backtest data. The data of the current day will be live data that we collected throughout the morning. If you make this history request in a backtest, you might get slightly different data for the current day because of post-processing from the data vendor.

In cloud deployments, QuantConnect is the default data provider. If you remove it and use a different data provider, you won't get any price data from QuantConnect. However, we provide the following datasets:

- US Equity Security Master
- US Futures Security Master
- US Equity Option Universe
- US Index Option Universe
- Universe selection datasets
- Non-streaming alternative datasets

QC Data Provider Limitations

When you request historical data or run warm-up , the amount of historical data you can access depends on the resolution of

your data subscriptions. The following table shows the amount of trailing historical data you can access for each data resolution:

Resolution	Available History
Daily	All historical data
Hour	All historical data
Minute	1 year
Second	2 months
Tick	1 month

In live trading, history requests for Index or Options with the QuantConnect data provider only returns data up to the end of the previous day. For example, if you make a history request for these asset classes at 1 PM Eastern Time (ET), you won't get any data for the current day.

For more information about the QuantConnect data provider, see QuantConnect .

Other Data Provider Limitations

LEAN supports other data providers, including IQFeed and Polygon. For more information about the limitations of these other data providers, see their respective documentation in Cloud Platform , Local Platform , or the CLI .

Brokerage Limitations

LEAN supports several brokerages that can provide historical data. However, some of them have limitations on historical data. For example, Interactive Brokers can be slow, so it can timeout if you request historical data for too many assets at once. For more information about these brokerages and the data they provide, see Datasets .

Historical Data

Live Trading

Introduction

When you make history requests in live trading, be aware of the associated data delays and the data provider's functionality.

Data Delays

Backtests have access to data up to $t - 1$ to $t - 4$ days, where t is the current day. The availability depends on the data type.

In live trading, history requests take 5-10 seconds to return the data. While LEAN processes the request, the algorithm doesn't receive any new data points.

Multiple Data Providers

When you deploy a live algorithm, you can add multiple data provider. If you use multiple data providers, the order you select them in defines their order of precedence in Lean. For example, if you set QuantConnect as the first provider and Interactive Brokers (IB) as the second provider, Lean only uses the IB data provider for securities that aren't available from the QuantConnect data provider. This configuration makes it possible to use the QuantConnect data provider for Equity universe selection and then place Options trades on the securities in the universe.

Data Provider Limitations

History requests from third-party data providers are much slower than from the QuantConnect data provider. If you need data from a third-party data provider, request small amounts or your algorithm may take longer than 10 minutes to process and cause a timeout. For more information about the data providers, see their respective documentation in Cloud Platform , Local Platform , or the CLI .

Trading and Orders

Trading and Orders

Key Concepts

Introduction

LEAN has dozens of methods to create, update, and cancel orders. You can place orders automatically with helper methods or manually through methods on the algorithm API. You can fetch, update, and cancel manual orders with order tickets . As the state of your orders change, LEAN creates events that notify your algorithm.

In backtesting, LEAN simulates order fills with historical data, but you can create your own fill, fee, slippage, and margin models via plugin points. You control how optimistic or pessimistic order fills are with transaction model classes. For more information about these types of models, see Reality Modeling .

Orders and Tickets

When you call one of the order methods to place an order, LEAN performs pre-order checks to ensure that the order meets some requirements and uses the last known price to check you have enough capital . To see the requirements of each order, see the Requirements section of the documentation for the order types and the Orders section of your brokerage model . If you can place the order, LEAN creates `Order` and OrderTicket objects. The order ticket is sent to your brokerage. As the brokerage processes your order, it returns another order ticket and it's compared against the order to see if the order is satisfied. Orders fill asynchronously in live trading, so if you want to change an order, you must request it with the order ticket. Order changes are not guaranteed since your order may fill by the time brokerage receives the request.

Order Types

LEAN supports the following order types:

- Market
- Limit
- Limit if touched
- Stop market
- Stop limit
- Trailing stop
- Market on open
- Market on close
- Combo market
- Combo limit
- Combo leg limit
- Option exercise

Symbol Properties

The SymbolProperties are a property of the Security object. LEAN uses some of the SymbolProperties to prevent invalid orders, and to calculate order quantities for a given target.

SymbolProperties objects have the following properties:

To get the SymbolProperties , use the property on the Security object.

```
C#

var symbolProperties = Securities["BTCUSD"].SymbolProperties;
var lotSize = symbolProperties.LotSize;
var minimumOrderSize = symbolProperties.MinimumOrderSize;
var minimumPriceVariation = symbolProperties.MinimumPriceVariation;
```

LEAN uses the MinimumPriceVariation to round the LimitPrice , StopPrice , and the TriggerPrice .

Quote Currency

The quote currency is the currency you must give the seller to buy an asset. For currency trades, the quote currency is the currency ticker on the right side of the currency pair. For other types of assets, the quote currency is usually USD, but the quote currency for India Equities is INR. To get the quote currency of a Security , use the QuoteCurrency property.

```
C#

var aaplQuoteCurrency = Securities["AAPL"].QuoteCurrency; // USD
var btcusdtQuoteCurrency = Securities["BTCUSDT"].QuoteCurrency; // USDT
```

The QuoteCurrency is a Cash object, which have the following properties:

You can use the ConversionRate property to calculate the value of the minimum price movement in the account currency

```
C#

var cfd = Securities["SG30SGD"];
var quoteCurrency = cfd.QuoteCurrency; // SGD
var contractMutiplier = cfd.SymbolProperties.ContractMultiplier;
var minimumPriceVariation = cfd.SymbolProperties.MinimumPriceVariation;

// Value of a pip in account currency
var pip = minimumPriceVariation * contractMutiplier * quoteCurrency.ConversionRate;
```

Trade Models

Reality models make backtests as realistic as possible to how the strategy would perform in live trading.

Split Adjustment

If a split event occurs before your order is filled, the unfilled portion of the order is adjusted automatically, where its quantity is multiplied by the split factor and the limit/stop/trigger price (if any) is divided by the split factor.

Open Order Margin Requirements

When an order fills, LEAN uses the portfolio information to check if you have enough capital to maintain the remaining open orders. If you don't, they are canceled.

The following table shows the order events of three orders of SPY 240109C00470000:

Time	Id	Type	Price	Quantity	Status
2024-01-08 09:31	1	Buy Market	0.91	20	`Filled`
	2	Sell Stop Market	0.81	-20	`Submitted`
2024-01-08 10:31	3	Sell Market	1.50	-10	`Filled`
	2	Sell Stop Market	0.81	-20	`Invalid`

When the sell market order fills at 10:31, LEAN checks the sell stop order and sets its status to `OrderStatus.Invalid` due to insufficient buying power to complete it.

Asynchronous Execution

In live trading, we send your orders to the API of your brokerage and wait for their response to update the status of the order ticket from `New` to `Submitted` , `Canceled` , or `Invalid` . When you place multiple orders, waiting for responses from the brokerage delays the last orders of the batch. To avoid this delay, you can place orders asynchronously. When you place an order asynchronously, LEAN sends the order to your brokerage and immediately continues executing your algorithm without waiting for a response. When the response from the brokerage arrives, LEAN updates the status of the order ticket accordingly, and your algorithm receives an order event .

Live Trading Considerations

In live trading, orders fill asynchronously. We send your order to the API of your brokerage and wait for their response to update the state of your algorithm and portfolio. The timing of live order fills doesn't depend on the resolution of your security subscriptions. When your order fills, the fill price and fee is set by your brokerage. You can add event handlers to your algorithm to monitor the brokerage connection and brokerage messages .

In backtesting, the trade fill timing depends on the resolution of your security subscription. For example, if you subscribe to a security with minute resolution data, the algorithm only receives data in one-minute time slices . As a result, the fill model can only evaluate if the order should fill on a minute-by-minute frequency.

Trading and Orders

Order Management

Order Management

Order Tickets

Introduction

When you create an order, you get an `OrderTicket` object to manage your order.

Properties

`OrderTicket` objects have the following attributes:

Track Orders

As the state of your order updates over time, your order ticket automatically updates. To track an order, you can check any of the preceding order ticket properties.

To get an order field, call the `Get` method with an `OrderField` .

```C#
private Symbol _symbol;
private OrderTicket _ticket;

public override void Initialize()
{
    _symbol = AddEquity("SPY").Symbol;
}

public override void OnData(Slice slice)
{
    // Place order if not invested and save the order ticket for later retrival.
    if (!Portfolio.Invested && slice.Bars.TryGetValue(_symbol, out var bar))
    {
        _ticket = LimitOrder(_symbol, 10, bar.Close);
    }
    // Get the limit price if the order is filled.
    else if (Portfolio.Invested)
    {
        var limitPrice = _ticket.Get(OrderField.LimitPrice);
    }
}
```

The `OrderField` enumeration has the following members:

In addition to using order tickets to track orders, you can receive order events through the `OnOrderEvent` event handler.

Update Orders

To update an order, use its `OrderTicket` . You can update other orders until they are filled or the brokerage prevents modifications. You just can't update orders during warm up and initialization .

Updatable Properties

The specific properties you can update depends on the order type. The following table shows the properties you can update for each order type.

Order Type	Updatable Properties				
	Tag	Quantity	LimitPrice	TriggerPrice	StopPrice
Market Order					
Limit Order	✓	✓	✓		
Limit If Touched Order	✓	✓	✓	✓	
Stop Market Order	✓	✓			✓
Stop Limit Order	✓	✓	✓		✓
Market On Open Order	✓	✓			
Market On Close Order	✓	✓			

Update Methods

To update an order, pass an `UpdateOrderFields` object to the `Update` method. The method returns an `OrderResponse` to signal the success or failure of the update request.

C#

```
private Symbol _symbol;
private OrderTicket _ticket;

public override void Initialize()
{
    _symbol = AddEquity("SPY").Symbol;
}

public override void OnData(Slice slice)
{
    // Place order if not order yet and save the order ticket for later retrival.
    if (_ticket == null && slice.Bars.TryGetValue(_symbol, out var bar))
    {
        _ticket = LimitOrder(_symbol, 10, bar.Close * 0.98m);
    }
    // If order is placed, update the limit price to be 90% of the orginal.
    else if (_ticket != null && _ticket.Status == OrderStatus.Submitted)
    {
        // Update the order tag and limit price
        var response = _ticket.Update(new UpdateOrderFields()
        {
            Tag = "Our New Tag for SPY Trade",
            LimitPrice = _ticket.Get(OrderField.LimitPrice * 0.9m)
        });

        // Check if the update request is successfully submitted to the broker.
        // Note that it may not represent the order is updated successfully: during the order updating process, it
may be filled or canceled.
        if (response.IsSuccess)
        {
            Debug("Order update request is submitted successfully");
        }
    }
}
```

To update individual fields of an order, call any of the following methods:

- `UpdateLimitPrice`

- `UpdateQuantity`
- `UpdateStopPrice`
- `UpdateTag`

```C#
var limitResponse = ticket.UpdateLimitPrice(limitPrice, tag);
var quantityResponse = ticket.UpdateQuantity(quantity, tag);
var stopResponse = ticket.UpdateStopPrice(stopPrice, tag);
var tagResponse = ticket.UpdateTag(tag);
```

Update Order Requests

When you update an order, LEAN creates an `UpdateOrderRequest` object, which have the following attributes:

To get a list of `UpdateOrderRequest` objects for an order, call the `UpdateRequests` method.

```C#
var updateRequests = ticket.UpdateRequests();
```

Workaround for Brokerages That Don't Support Updates

Not all brokerages fully support order updates. To check what functionality your brokerage supports for order updates, see the Orders section of the documentation for your brokerage model . If your brokerage doesn't support order updates and you want to update an order, cancel the order . When you get an order event that confirms the order is no longer active, place a new order.

```C#
private Symbol _symbol;
private OrderTicket _ticket;

public override void Initialize()
{
    _symbol = AddEquity("SPY").Symbol;
}

public override void OnData(Slice slice)
{
    // Place order if not order yet and save the order ticket for later retrival.
    if (_ticket == null && slice.Bars.TryGetValue(_symbol, out var bar))
    {
        _ticket = LimitOrder(_symbol, 10, bar.Close);
    }
    // If order is placed, cancel the order and place a new one as substituent.
    else if (_ticket != null && _ticket.Status == OrderStatus.Submitted)
    {
        // Cancel the order
        _ticket.Cancel();
    }
}

public override void OnOrderEvent(OrderEvent orderEvent)
{
    if (_ticket != null
        && orderEvent.OrderId == _ticket.OrderId
        && orderEvent.Status == OrderStatus.Canceled)
    {
        // Place a new order
        var quantity = _ticket.Quantity - _ticket.QuantityFilled;
        var limitPrice = Securities[_ticket.Symbol].Price + 1;
        _ticket = LimitOrder(_ticket.Symbol, quantity, limitPrice);
    }
}
```

Cancel Orders

To cancel an order, call the `Cancel` method on the `OrderTicket` . The method returns an `OrderResponse` object to signal the success or failure of the cancel request.

```C#
private Symbol _symbol;
private OrderTicket _ticket;

public override void Initialize()
{
    _symbol = AddEquity("SPY").Symbol;
}

public override void OnData(Slice slice)
{
    // Place order if not order yet and save the order ticket for later retrival.
    if (_ticket == null && slice.Bars.TryGetValue(_symbol, out var bar))
    {
        _ticket = LimitOrder(_symbol, 10, bar.Close);
    }
    // If order is placed, cancel the order if it is not filled within 2 minutes.
    else if (_ticket != null && _ticket.Time <= slice.Time.AddMinutes(-2) && _ticket.Status == OrderStatus.Submitted)
    {
        // Cancel the order
        var response = ticket.Cancel("Canceled SPY trade");

        // Check if the cancel request is successfully submitted to the broker.
        // Note that it may not represent the order is canceled successfully: during the order updating process, it
may be filled.
        if (response.IsSuccess)
        {
            Debug("Order cancel request successfully submitted");
        }
    }
}
```

You can't cancel market orders because they are immediately transmitted to the brokerage. You also can't cancel any orders during warm up and initialization .

When you cancel an order, LEAN creates a `CancelOrderRequest` , which have the following attributes:

To get the `CancelOrderRequest` for an order, call the `CancelRequest` method. The method returns `null` if the order hasn't been cancelled.

```
C#
var request = ticket.cancel_order_request();
```

Order Response

When you update or cancel an order, LEAN returns an `OrderReponse` object, which have the following attributes:

If your order changes fail, check the `ErrorCode` or `ErrorMessage` . For more information about specific order errors, see the Order Response Error Reference .

To get most recent order response, call the `GetMostRecentOrderResponse` method.

```
C#
var response = ticket.get_most_recent_order_response();
```

Examples

The following examples demonstrate some common practices for order ticket management.

Example 1: Timed Exit

The following algorithm is an EMA-crossover strategy. Using the filled time of the order ticket, we can control to exit the position after 45 minutes.

C#

```
public class OrderTicketExampleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ExponentialMovingAverage _ema;
    private OrderTicket _ticket;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        _symbol = AddEquity("SPY").Symbol;
        // Set up EMA indicator for trade signal generation.
        _ema = EMA(_symbol, 20, Resolution.Daily);
        // Warm up the indicator for its readiness to use immediately.
        WarmUpIndicator(_symbol, _ema);
    }

    public override void OnData(Slice slice)
    {
        // Place order if not order yet and save the order ticket for later retrival.
        if (_ticket == null && slice.Bars.TryGetValue(_symbol, out var bar))
        {
            // EMA crossover: buy if price is above EMA, which indicate uptrend.
            if (_ema < bar.Close)
            {
                _ticket = LimitOrder(_symbol, 100, bar.Close * 0.995m);
            }
            // EMA crossover: sell if price is below EMA, which indicate down trend.
            else
            {
                _ticket = LimitOrder(_symbol, -100, bar.Close * 1.005m);
            }
        }
        // Exit position if it is in the portfolio for more than 45 minutes.
        else if (_ticket != null && _ticket.Time <= UtcTime.AddMinutes(-45))
        {
            Liquidate();
            _ticket = null;
        }
    }
}
```

Example 2: Intraday Stop Loss

The following algorithm demonstrates an intraday long position with a 10% stop loss order. Before the market closes, cancel the stop loss order if it is not filled and exit the position at market close with a market on close order.

```C#
public class OrderTicketExampleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    // A variable to save the stop loss order and track if it is filled.
    private OrderTicket _ticket;
    private int _day = -1;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        _symbol = AddEquity("QQQ").Symbol;

        // Schedule an event to handle exiting position before market close if stop loss is not hit.
        // Market on close order requires ordering 15 minutes before market close.
        Schedule.On(
            DateRules.EveryDay(_symbol),
            TimeRules.BeforeMarketClose(_symbol, 16),
            ExitPosition
        );
    }

    public override void OnData(Slice slice)
    {
        // Place order if not invested. Make sure only trade once a day to avoid chasing the loss.
        if (!Portfolio.Invested && _day != slice.Time.Day && slice.Bars.TryGetValue(_symbol, out var bar))
        {
            MarketOrder(_symbol, 10);
            // Update the day variable to avoid repeat ordering.
            _day = slice.Time.Day;
        }
    }

    public override void OnOrderEvent(OrderEvent orderEvent)
    {
        // Make sure to order the stop loss order on the filled market order only.
        if (orderEvent.Status == OrderStatus.Filled && orderEvent.FillQuantity > 0)
        {
            // Place a 10% stop loss order to avoid large loss. Save the order ticket for tracking its status later.
            _ticket = StopMarketOrder(orderEvent.Symbol, -orderEvent.FillQuantity, orderEvent.FillPrice*0.9m);
        }
    }

    private void ExitPosition()
    {
        // Only need to cancel the order and handle exit position if the stop loss order is not fully filled.
        if (_ticket != null && _ticket.Status != OrderStatus.Filled)
        {
            _ticket.Cancel();
            // Market on close order to exit position at market close with the remaining quantity.
            MarketOnCloseOrder(_symbol, -10 - _ticket.QuantityFilled);
        }
    }
}
```

Example 3: Crypto Trailing Stop Order

This example trades BTCUSD with a similar logic of Example 2, but apply a trailing stop loss of 10% instead. Crypto exchanges does not support trailing stop loss order. Yet, we can cancel and reorder a stop loss order with the updated stop loss price. If the market price of the crypto pair is above the previous level, update the stop loss order by the above.

```C#
public class OrderTicketExampleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    // A variable to hold the stop loss order to track if it should be updated.
    private OrderTicket _ticket;
    // A day variable to avoid over-ordering (trade once per day)
    private int _day = -1;
    // A variable to hold the price of the high price in the lifetime of the open position for updating the stop loss
price.
    private decimal _highPrice = 0m;
    // Update the stop loss when new high is 1% higher
    private decimal _updateStep = 1.01m;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Set the market argument to trade in specific exchange.
        _symbol = AddCrypto("BTCUSD", market: Market.Coinbase).Symbol;
    }

    public override void OnData(Slice slice)
    {
        if (!slice.Bars.TryGetValue(_symbol, out var bar)) return;

        // Place order if not invested. Make sure only trade once a day to avoid chasing the loss.
        if (!Portfolio.Invested && _day != Time.Day)
        {
            MarketOrder(_symbol, 1);
            // Place a 10% stop loss order to avoid large loss. Save the order ticket for tracking its status later.
            _ticket = StopMarketOrder(_symbol, -1, bar.Close*0.9m);
            // Update the day variable to avoid repeat ordering.
            _day = Time.Day;
        }

        // Update the trailing stop order. Since order updating is not supported in crypto exchanges, we cancel and
reorder.
        // Trailing stop price will only need to be updated when the market price is higher than the current price.
        if (_ticket != null &&_ticket.Status.IsOpen() && bar.High > _highPrice)
        {
            _ticket.Cancel();
            _ticket = StopMarketOrder(_symbol, -Portfolio[_symbol].Quantity, bar.High*0.9m);
            _highPrice = bar.High * _updateStep;
        }
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
OrderTicketDemoAlgorithm.cs C#

Order Management

Transaction Manager

Introduction

The algorithm transactions manager (`SecurityTransactionManager`) contains a collection of helper methods to quickly access all your orders. To access the transaction manager, use the `Transactions` property of your algorithm. If you save a reference to your order tickets , you shouldn't need to use the transaction manager. LEAN updates the order tickets as the brokerage processes your orders.

Get a Single Order Ticket

If you didn't save a reference to the order ticket when you created an order, you can call the `GetOrderTicket` method to get it. You need to pass the order ID to the method. If you don't have the order ID, you can use the `LastOrderId` property to get the order ID of the most recent order.

```
C#
var orderId = Transactions.LastOrderId;
var ticket = Transactions.GetOrderTicket(orderId);
```

Get Order Tickets

To get order tickets, call the `GetOrderTickets` or `GetOpenOrderTickets` method. You can pass in a filter function to filter all of the order tickets or pass a `Symbol` to get the order tickets for a specific asset.

```
C#
// Get all order tickets
var orderTickets = Transactions.GetOrderTickets();

// Get order tickets that pass a filter
var filteredOrderTickets = Transactions.GetOrderTickets(orderTicket => orderTicket.Symbol == symbol);

// Get all open order tickets
var openOrderTickets = Transactions.GetOpenOrderTickets();

// Get all open order tickets for a symbol
var symbolOpenOrderTickets = Transactions.GetOpenOrderTickets(symbol);

// Get open order tickets that pass a filter
var filteredOpenOrderTickets = Transactions.GetOpenOrderTickets(orderTicket => orderTicket.Quantity > 10);
```

Get a Single Order

To get a clone of a specific order, call the `GetOrderById` method with the order Id. To get the order Id, use the `OrderId` property of the order ticket or use the `LastOrderID` property if you want the most recent order.

```
C#
var orderId = Transactions.LastOrderID;
var order = Transactions.GetOrderById(orderId);
```

Order objects are immutable and changes to the order object will not impact the trade. To make an update to an order you must use Order Tickets . `Order` objects have the following attributes:

Get Orders

To get a list of orders, call the `GetOrders` , `GetOpenOrders` , or `GetOrdersByBrokerageId` method. These method returns a list of `Order` objects.

```
C#

// Get all completed orders
var completedOrders = Transactions.GetOrders();

// Get all completed orders that pass a filter
var filteredCompletedOrders = Transactions.GetOrders(x => x.Quantity > 10);

// Get a list of all completed orders for a symbol
var symbolCompletedOrders = Transactions.GetOrders(x => x.Symbol == symbol);

// Get all open orders
var openOrders = Transactions.GetOpenOrders();

// Get all open orders that pass a filter
var filteredOpenOrders = Transactions.GetOpenOrders(x => x.Quantity > 10);

// Get a list of all open orders for a symbol
var symbolOpenOrders = Transactions.GetOpenOrders(symbol);

// Get all open and completed orders that correspond to an Id that the brokerage assigned in live trading
var ordersByBrokerageId = Transactions.GetOrdersByBrokerageId(brokerageId);
```

`Order` objects have the following attributes:

The `OrdersCount` property gets the current number of orders that have been processed.

Get Remaining Order Quantity

To get the unfilled quantity of open orders, call the `GetOpenOrdersRemainingQuantity` method.

```
C#

// Get the quantity of all open orders
var allOpenQuantity = Transactions.GetOpenOrdersRemainingQuantity();

// Get the quantity of open orders that pass a filter
var filteredOpenQuantity = Transactions.GetOpenOrdersRemainingQuantity(
    orderTicket => orderTicket.Quantity > 10
);

// Get the quantity of open orders for a symbol
var symbolOpenQuantity = Transactions.GetOpenOrdersRemainingQuantity(symbol);
```

Cancel Orders

To cancel open orders, call the `CancelOpenOrders` method. This method returns a list of `OrderTicket` objects that correspond to the canceled orders.

```
C#

// Cancel all open orders
var allCancelledOrders = Transactions.CancelOpenOrders();

// Cancel orders related to IBM and apply a tag
var ibmCancelledOrders = Transactions.CancelOpenOrders("IBM", "Hit stop price");
```

Live Trading Considerations

LEAN doesn't save orders and order tickets across live deployments.

When you deploy your algorithm, LEAN fetches the open orders from your brokerage and populate the algorithm transactions

manager (`SecurityTransactionManager`).

Examples

The following examples demonstrate some common practices for using the transaction manager.

Example 1: 2-Level Take Profit And Stop Loss

The following algorithm trades volatility using the CBOE VIX dataset . It makes use of the transaction manager to handle a 2-layer take profit and stop loss risk management.

```
C#
public class TransactionManagerAlgorithm : QCAlgorithm
{
    private Symbol _spy, _vix;
    private ExponentialMovingAverage _ema;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request SPY data for trading and trade on daily signals so we use daily resolution.
        _spy = AddEquity("SPY", Resolution.Daily).Symbol;
        // Subscribe to VIX data to generate trade signals.
        _vix = AddData<CBOE>("VIX", Resolution.Daily).Symbol;

        // Use the EMA indicator to generate trade signals from VIX.
        _ema = EMA(_vix, 20, Resolution.Daily);
        // Warm up the EMA indicator to enable immediate use.
        WarmUpIndicator(_vix, _ema, Resolution.Daily);
    }

    public override void OnData(Slice slice)
    {
        if (slice.Bars.TryGetValue(_spy, out var spyBar))
        {
            var vix = Securities[_vix].Price;

            // Trade based on volatility. Sell SPY if VIX is in an uptrend, indicating low market volatility.
            // Buy SPY if VIX is in a downtrend, indicating a volatile market.
            if (vix > _ema && !Portfolio[_spy].IsLong)
            {
                // Cancel all open orders to remove redundant take profit and stop loss orders.
                Transactions.CancelOpenOrders();

                var quantity = 10 - Portfolio[_spy].Quantity;
                MarketOrder(_spy, quantity);
                // Stop Loss at 2%.
                StopMarketOrder(_spy, -10, spyBar.Close * 0.98m, tag: "stop loss");
                // Take profit at 4%.
                LimitOrder(_spy, -5, spyBar.Close * 1.04m, tag: "take profit");
            }
            else if (vix < _ema && !Portfolio[_spy].IsShort)
            {
                // Cancel all open orders to remove redundant take profit and stop loss orders.
                Transactions.CancelOpenOrders();

                var quantity = -10 - Portfolio[_spy].Quantity;
                MarketOrder(_spy, quantity);
                // Stop Loss at 2%.
                StopMarketOrder(_spy, 10, spyBar.Close * 1.02m, tag: "stop loss");
                // Take profit at 4%.
                LimitOrder(_spy, 5, spyBar.Close * 0.96m, tag: "take profit");
            }
        }
    }

    public override void OnOrderEvent(OrderEvent orderEvent)
    {
        // If the take profit order was filled out, we place the 2nd layer of take profit.
        var takeProfitOrders = Transactions.GetOrderTickets(x => x.OrderType == OrderType.Limit).ToList();
        if (takeProfitOrders.Count == 0)
        {
            return;
        }
        var lastTakeProfitOrderId = takeProfitOrders.MaxBy(x => x.Time).OrderId;
```

```
        var lastTakeProfitOrderId = takeProfitOrders.MaxBy(x => x.Time).OrderId;

        if (orderEvent.Status == OrderStatus.Filled && orderEvent.Id == lastTakeProfitOrderId)
        {
            // Cancel all open orders to remove redundant stop loss orders.
            Transactions.CancelOpenOrders();
            // 2nd layer Take profit at extra 4%.
            var takeProfitPrice = orderEvent.FillQuantity > 0 ? orderEvent.FillPrice * 0.96m : orderEvent.FillPrice
* 1.04m;
            LimitOrder(_spy, 5, takeProfitPrice, tag: "second-layer take profit");
        }
        else if (orderEvent.Status == OrderStatus.Filled && orderEvent.Ticket.OrderType == OrderType.StopMarket)
        {
            // Cancel all open orders to remove redundant take profit orders if stop loss.
            Transactions.CancelOpenOrders();
        }
    }
}
```

Trading and Orders

Order Types

You can place any of the following order types that your brokerage supports. Click one to learn more.

Market Orders

Limit Orders

Limit if Touched Orders

Stop Market Orders

Stop Limit Orders

Trailing Stop Orders

Market On Open Orders

Market On Close Orders

Combo Market Orders

Combo Limit Orders

Combo Leg Limit Orders

Option Exercise Orders

Other Order Types

See Also

Brokerages

Order Types

Market Orders

Introduction

Market orders are instructions to trade a specific number of units, regardless of the fill price. If your highest priority is to fill an order as fast as possible, use market orders. If the order book of the security you're trading has sufficient liquidity when the brokerage receives your order, it immediately fills. However, if there is not enough liquidity by the brokerage, you may need to wait a few minutes to fill the order, you may get partial fills, and you may incur additional costs from market impact.

Place Orders

To send a market order, call the `MarketOrder` , `Buy` , `Sell` , or `Order` method and provide a `Symbol` and quantity. If you don't have sufficient capital for the order, it's rejected. By default, market orders are synchronous and fill immediately.

```
C#
// Buy orders
MarketOrder("IBM", 100);
Buy("AAPL", 10);
Order("SPY", 20);

// Sell orders
MarketOrder("AMZN", -15);
Sell("TSLA", 25);
Order("SPY", -20);
```

You can also provide a tag and order properties to the `Order` and `MarketOrder` methods.

```
C#
MarketOrder(symbol, quantity, tag: tag, orderProperties: orderProperties);
```

If you place a market order during pre-market or post-market hours, LEAN converts your order to market on open order. To override this behavior, create a custom fill model . To be notified when the default fill model allows you to submit market orders outside of regular trading hours, subscribe to GitHub Issue #3947 .

Monitor Order Fills

If the brokerage has sufficient liquidity in their order book, market orders fill immediately. Otherwise, you get partial fills. To monitor the fills of your order, save a reference to the order ticket .

```
C#
var ticket = MarketOrder("XLK", 10);
Debug($"Quantity filled: {ticket.QuantityFilled}; Fill price: {ticket.AverageFillPrice}");
```

For more information about how LEAN models order fills in backtests, see Trade Fills .

Cancel Orders

To cancel a market order, call the `Cancel` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Cancel` method returns an `OrderResponse` object to signal the success or failure of the cancel request.

```
var response = ticket.Cancel("Cancelled trade");
if (response.IsSuccess)
{
    Debug("Order successfully cancelled");
}
```

When you cancel an order, LEAN creates a `CancelOrderRequest` , which have the following attributes:

To get the `CancelOrderRequest` for an order, call the `CancelRequest` method on the order ticket. The method returns `null` if the order hasn't been cancelled.

```
var request = ticket.cancel_order_request();
```

Asynchronous Orders

When you trade a large portfolio of assets, you may want to send orders in batches and not wait for the response of each one. To send asynchronous orders , set the `asynchronous` argument to `true` .

```
MarketOrder("IBM", 100, asynchronous: true);
```

Synchronous Timeouts

Market orders are synchronous by default, so your algorithm waits for the order to fill before moving to the next line of code. If your order takes longer than five seconds to fill, your algorithm continues executing even if the trade isn't filled. To adjust the timeout period, set the `Transactions.MarketOrderFillTimeout` property.

```
// Adjust the market fill-timeout to 30 seconds.
Transactions.MarketOrderFillTimeout = TimeSpan.FromSeconds(30);
```

Market orders may take a few minutes to fill for illiquid assets such as out-of-the-money Options or penny stocks.

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with market orders, set the brokerage model to a brokerage that supports them.

```
SetBrokerageModel(BrokerageName.QuantConnectBrokerage);
```

To check if your brokerage has any special requirements for market orders, see the Orders section of the brokerage model documentation .

Requirements

You can submit market orders during regular trading hours for all asset classes. For Futures and Future Options, you can place market orders during regular and extended market hours. For all other asset classes, if you place market orders at/after the last minute of regular market hours, LEAN converts the market orders into market-on-open orders as long as the asset class supports them . To view the trading hours of each asset class, follow these steps:

1. Open the Asset Classes documentation.
2. Click an asset class.
3. Click Market Hours .

Examples

The following backtest verifies the `MarketOrder` behavior. The algorithm buys SPY on the first day and liquidates the position on the second day. The following table shows the first two trades in the backtest:

Time	Symbol	Price	Quantity	Type	Status	Value	Tag
2021-07-01T09:31:00Z	SPY	429.11	10	Market	Filled	4291.10	
2021-07-02T09:31:00Z	SPY	431.54	-10	Market	Filled	-4315.40	

On July 1, 2021, the algorithm buys SPY at $429.11. The fill model fills this order at the ask close price.

On July 2, 2021, the algorithm sells the SPY holdings at $431.54. The fill model fills this order at the bid close price.

To reproduce these results, backtest the following algorithm:

C#

```
public class MarketOrderAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 4);
        SetCash(100000);
        UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;
        _symbol = AddEquity("SPY").Symbol;
    }

    public override void OnData(Slice data)
    {
        if (!Portfolio.Invested && Time.Day == 1)
        {
            MarketOrder(_symbol, 10);
        }
        else if (Portfolio.Invested && Time.Day == 2)
        {
            MarketOrder(_symbol, -10);
        }
    }
}
```

Order Types

Limit Orders

Introduction

Limit orders are instructions to trade a quantity of an asset at a specific price or a better price. A marketable limit order has a limit price that crosses the spread. An unmarketable limit order has a limit price that doesn't cross the spread. For example, if an asset has an ask price of 100*andyouplacealimitordertobuyat*100 or higher, that's a marketable limit order that should immediately fill. If you place a limit order to buy at $99.99 or lower, that's an unmarketable limit order that should go on the order book. Unmarketable limit orders save you from paying spread costs. Some brokerages even charge you a lower transaction fee if you use unmarketable limit orders rather than marketable limit orders.

Limit orders are helpful in illiquid markets. You can use them to get a good entry price or to set a take-profit level on an existing holding. However, if the market trades away from your limit price, you may have to adjust your limit price or wait for the market to trade back to your limit price to fill the order.

Place Orders

To send a limit order, call the `LimitOrder` method with a `Symbol` , quantity, and limit price. You can also provide a tag and order properties to the `LimitOrder` method. If you do not have sufficient capital for the order, it's rejected.

```
C#
LimitOrder(symbol, quantity, limitPrice, tag, orderProperties);
```

To buy an asset with a marketable limit order, set the limit price to the current ask price or higher.

To sell an asset with a marketable limit order, set the limit price to the current bid price or lower.

To buy an asset with an unmarketable limit order, set the limit price below the current ask price.

```
C#
// Buy 1 Bitcoin when the price drops to $34,000
LimitOrder("BTCUSD", 1, 34000);
```

To sell an asset with an unmarketable limit order, set the limit price above the current bid price.

```C#
// Sell 1 Bitcoin when the price moves up to $60,000
LimitOrder("BTCUSD", -1, 60000);
```

Monitor Order Fills

Limit orders fill when the security price passes the limit price. To monitor the fills of your order, save a reference to the order ticket .

```C#
// Buy 10 shares of XLK at $140
var ticket = LimitOrder("XLK", 10, 140);
Debug($"Quantity filled: {ticket.QuantityFilled}; Fill price: {ticket.AverageFillPrice}");
```

For more information about how LEAN models order fills in backtests, see Trade Fills .

Update Orders

You can update the quantity, limit price, and tag of limit orders until the order fills or the brokerage prevents modifications. To

update an order, pass an `UpdateOrderFields` object to the `Update` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Update` method returns an OrderResponse to signal the success or failure of the update request.

```C#
// Create a new order and save the order ticket
var ticket = LimitOrder("SPY", 100, 221.05m, tag: "original tag");

// Update the order
var response = ticket.Update(new UpdateOrderFields()
{
    Quantity = 80,
    LimitPrice = 222.00m,
    Tag = "new tag"
});

// Check if the update was successful
if (response.IsSuccess)
{
    Debug("Order updated successfully");
}
```

To update individual fields of an order, call any of the following methods:

- `UpdateLimitPrice`
- `UpdateQuantity`
- `UpdateTag`

```C#
var limitResponse = ticket.UpdateLimitPrice(limitPrice, tag);

var quantityResponse = ticket.UpdateQuantity(quantity, tag);

var tagResponse = ticket.UpdateTag(tag);
```

When you update an order, LEAN creates an `UpdateOrderRequest` object, which have the following attributes:

To get a list of `UpdateOrderRequest` objects for an order, call the `UpdateRequests` method.

```C#
var updateRequests = ticket.UpdateRequests();
```

Cancel Orders

To cancel a limit order, call the `Cancel` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Cancel` method returns an `OrderResponse` object to signal the success or failure of the cancel request.

```C#
var response = ticket.Cancel("Cancelled trade");
if (response.IsSuccess)
{
    Debug("Order successfully cancelled");
}
```

When you cancel an order, LEAN creates a `CancelOrderRequest` , which have the following attributes:

To get the `CancelOrderRequest` for an order, call the `CancelRequest` method on the order ticket. The method returns `null` if the order hasn't been cancelled.

```
var request = ticket.cancel_order_request();
```

Asynchronous Orders

When you trade a large portfolio of assets, you may want to send orders in batches and not wait for the response of each one. To send asynchronous orders , set the `asynchronous` argument to `true` .

```
LimitOrder("SPY", 10, 429.0m, asynchronous: true);
```

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with limit orders, set the brokerage model to a brokerage that supports them.

```
SetBrokerageModel(BrokerageName.QuantConnectBrokerage);
```

To check if your brokerage has any special requirements for limit orders, see the Orders section of the brokerage model documentation .

Requirements

Limit orders can be submitted at any time for all security types.

If your algorithm subscribes to extended market hours, they can be filled outside regular trading hours.

Examples

The following backtest verifies the `LimitOrder` method behavior. On the first day, the algorithm buys SPY with an unmarketable limit order. On the second day, the algorithm sells SPY with an unmarketable limit order and places another unmarketable limit order to buy SPY, which doesn't fill. The following table shows the first three trades in the backtest:

Submitted Time	Filled Time	Symbol	Limit Price	Filled Price	Quantity	Type	Status	Value	Tag
2021-07-01T09:31:00Z	2021-07-01T09:32:00Z	SPY	429.00	429.00	10	Limit	Filled	4290.00	Limit Price: ¤429.00
2021-07-02T09:31:00Z	2021-07-02T09:35:00Z	SPY	431.70	431.70	-10	Limit	Filled	-4317.00	Limit Price: ¤431.70
2021-07-02T09:31:00Z	/	SPY	400.00	/	10	Limit	Submitted	/	Limit Price: ¤400.00

On July 1, 2021 at 9:31 AM Eastern Time (ET), the algorithm places a buy limit order with a limit price of 429*whentheasklowpriceis* 428.81 and the ask high price is 429.15.*Theorderfillsat*9:32*AMETatapriceof*429. The fill model fills buy limit orders when the ask low price is less than the limit price. It sets the fill price of the order to the minimum of ask high price and the limit price.

On July 2, 2021 at 9:31 AM ET, the algorithm places a sell limit order at 431.70*andabuylimitorderat*400. At the time of the order, the bid high price is 431.65, *thebidlowpriceis*431.49, and the ask low price is 431.50.*TheselllimitorderfillsatA*9:35*AMETatapriceof*431.70

and the buy limit order doesn't fill. The fill model fills sell limit order when the bid high price is greater than the limit price. It sets the fill price of the order to the maximum of the bid low price and the limit price. The buy limit order doesn't fill because the ask low price is above the limit price for the remainder of the backtest period.

To reproduce these results, backtest the following algorithm:

```C#
public class LimitOrderAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 4);
        SetCash(100000);
        UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;
        _symbol = AddEquity("SPY").Symbol;
    }

    public override void OnData(Slice data)
    {
        if (Time.Day == 1 && Time.Hour == 9 && Time.Minute == 31)
        {
            LimitOrder(_symbol, 10, 429.0m);
        }
        else if (Time.Day == 2 && Time.Hour == 9 && Time.Minute == 31)
        {
            LimitOrder(_symbol, -10, 431.7m);
            LimitOrder(_symbol, 10, 400.0m);
        }
    }
}
```

Order Types

Limit if Touched Orders

Introduction

Limit if touched (LIT) orders are instructions to place a limit order once an asset touches a specific price level. A buy LIT order has a trigger price below the current market price and a sell LIT order has a trigger price above the current market price. In effect, LIT orders are the opposite of stop limit orders .

Place Orders

To send a LIT order, call the `LimitIfTouchedOrder` method and provide a `Symbol` , quantity, trigger price, and limit price. You can also provide a tag and order properties to the `LimitIfTouchedOrder` method. If you do not have sufficient capital for the order, it's rejected.

```
C#
LimitIfTouchedOrder(symbol, quantity, triggerPrice, limitPrice, tag, orderProperties);
```

To buy an asset with a LIT order that has a marketable limit price, set the trigger price below the current market price and set the limit price above the trigger price.

```
C#
// Once Bitcoin trades down to $36,000, place a limit order to buy 1 Bitcoin at $38,000
LimitIfTouchedOrder("BTCUSD", 1, 33000, 36000);
```

To sell an asset with a LIT order that has a marketable limit price, set the trigger price above the current market price and set the limit price below the trigger price.

```
C#
// Once Bitcoin trades up to $62,000, place a limit order to sell 1 Bitcoin at $59,000
LimitIfTouchedOrder("BTCUSD", -1, 62000, 59000);
```

To buy an asset with a LIT order that has an unmarketable limit price, set the trigger price below the current market price and set the limit price below the trigger price.

```C#
// Once Bitcoin trades down to $46,000, place a limit order to buy 1 Bitcoin at $36,000
LimitIfTouchedOrder("BTCUSD", 1, 46000, 36000);
```

To sell an asset with a LIT order that has an unmarketable limit price, set the trigger price above the current market price and set the limit price above the trigger price.

```C#
// Once Bitcoin trades up to $54,000, place a limit order to sell 1 Bitcoin at $62,000
LimitIfTouchedOrder("BTCUSD", -1, 54000, 62000);
```

Monitor Order Fills

LIT orders fill when the security price touches the trigger price and is at least as favorable as the limit price. To monitor the fills of your order, save a reference to the order ticket .

```
// Once SPY trades down to $425, place a limit order to buy 10 shares at $415
var ticket = LimitIfTouchedOrder("SPY", 10, 425, 415);
Debug($"Quantity filled: {ticket.QuantityFilled}; Fill price: {ticket.AverageFillPrice}");
```

For more information about how LEAN models order fills in backtests, see Trade Fills .

Update Orders

You can update the quantity, trigger price, limit price, and tag of LIT orders until the order fills or the brokerage prevents modifications. To update an order, pass an `UpdateOrderFields` object to the `Update` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Update` method returns an OrderResponse to signal the success or failure of the update request.

```
// Create a new order and save the order ticket
var ticket = LimitIfTouchedOrder("SPY", 100, 350, 340, tag: "Original tag");

// Update the order
var updateOrderFields = new UpdateOrderFields()
{
    Quantity = 80,
    TriggerPrice = 380,
    LimitPrice = 370,
    Tag = "New tag"
}
var response = ticket.Update(updateOrderFields);

// Check the OrderResponse
if (response.IsSuccess)
{
    Debug("Order updated successfully");
}
```

To update individual fields of an order, call any of the following methods:

- `UpdateLimitPrice`

- `UpdateQuantity`
- `UpdateTriggerPrice`
- `UpdateTag`

```
C#
var limitResponse = ticket.UpdateLimitPrice(limitPrice, tag);

var quantityResponse = ticket.UpdateQuantity(quantity, tag);

var triggerResponse = ticket.UpdateTriggerPrice(triggerPrice, tag);

var tagResponse = ticket.UpdateTag(tag);
```

When you update an order, LEAN creates an `UpdateOrderRequest` object, which have the following attributes:

To get a list of `UpdateOrderRequest` objects for an order, call the `UpdateRequests` method.

```
C#
var updateRequests = ticket.UpdateRequests();
```

Cancel Orders

To cancel a LIT order, call the `Cancel` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Cancel` method returns an `OrderResponse` object to signal the success or failure of the cancel request.

```
C#
var response = ticket.Cancel("Cancelled trade");
if (response.IsSuccess)
{
    Debug("Order successfully cancelled");
}
```

When you cancel an order, LEAN creates a `CancelOrderRequest` , which have the following attributes:

To get the `CancelOrderRequest` for an order, call the `CancelRequest` method on the order ticket. The method returns `null` if the order hasn't been cancelled.

```
C#
var request = ticket.cancel_order_request();
```

Asynchronous Orders

When you trade a large portfolio of assets, you may want to send orders in batches and not wait for the response of each one. To send asynchronous orders , set the `asynchronous` argument to `true` .

```
C#
LimitIfTouchedOrder("SPY", 10, 429.0m, 428.95m, asynchronous: true);
```

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with LIT orders, set the brokerage model to a brokerage that supports them.

```
SetBrokerageModel(BrokerageName.QuantConnectBrokerage);
```

To check if your brokerage has any special requirements for LIT orders, see the Orders section of the brokerage model documentation .

Requirements

LIT orders can be submitted at any time for all security types.

If your algorithm subscribes to extended market hours, they can be filled outside regular trading hours.

Examples

The following backtest verifies the `LimitIfTouchedOrder` behavior. The following table shows the trades in the backtest:

Submitted Time	Filled Time	Symbol	Trigger Price	Limit Price	Filled Price	Quantity	Type	Status	Value	Tag
2021-07-01T09:31:00Z	2021-07-01T09:38:00Z	SPY	429.00	428.95	428.95	10	Limit if touched	Filled	4285.00	Trigger Price: ¤429.00 Limit Price: ¤428.95
2021-07-02T09:31:00Z	2021-07-02T09:49:00Z	SPY	431.70	431.75	431.84	-10	Limit if touched	Filled	-4318.40	Trigger Price: ¤431.70 Limit Price: ¤431.75
2021-07-02T09:31:00Z	/	SPY	431.70	400.00	/	10	Limit if touched	Submitted	/	Trigger Price: ¤431.70 Limit Price: ¤400.00
/	/	SPY	400.00	400.00	/	10	Limit if touched	Submitted	/	Trigger Price: ¤400.00 Limit Price: ¤400.00

On July 1, 2021 at 9:31 AM Eastern Time (ET), the algorithm places a LIT order to buy SPY with a trigger price of 429*and a limit price of* 428.95. At the time of the order submission, the low price was 428.80 *and the bid close price was* 429.10. The order fills at 9:38 AM ET at a price of $428.95. The fill model sets the limit price for buy orders when the low price is less than or equal to the trigger price, fills the order when the bid close price is less than or equal to the limit price, and sets the fill price to the minimum of the ask price and the limit price.

On July 2, 2021 at 9:31 AM ET, the algorithm places a LIT order to sell SPY with a trigger price of 431.70 *and a limit price of* 431.75. At the time of the order submission, the high price was 431.78 *and the ask close price was* 431.55. The order fills at 9:49 AM ET at a price of $431.84. The fill model sets the limit price for sell orders when the high price is greater than or equal to the trigger price, fills the order when the ask close price is greater than or equal to the limit price, and sets the fill price to the maximum of the bid price and the limit price.

On July 2, 2021, the algorithm places a third LIT order to buy SPY with a trigger price of $431.70 and a limit price of $400. The fill model sets the fill price, but it doesn't fill the order because the limit price is too low. On the same day, the algorithm places a fourth LIT order to buy SPY with a trigger price of $400 and a limit price of $400. The fill model doesn't set the limit price for this order because SPY doesn't touch the trigger price before the backtest ends.

To reproduce these results, backtest the following algorithm:

```C#
public class LimitOrderAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 4);
        SetCash(100000);
        UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;
        _symbol = AddEquity("SPY").Symbol;
    }

    public override void OnData(Slice data)
    {
        if (Time.Day == 1 && Time.Hour == 9 && Time.Minute == 31)
        {
            LimitIfTouchedOrder(_symbol, 10, 429.0m, 428.95m);
        }
        else if (Time.Day == 2 && Time.Hour == 9 && Time.Minute == 31)
        {
            LimitIfTouchedOrder(_symbol, -10, 431.7m, 431.75m);
            LimitIfTouchedOrder(_symbol, 10, 431.7m, 400.0m);
            LimitIfTouchedOrder(_symbol, 10, 400.0m, 400.0m);
        }
    }
}
```

Order Types

Stop Market Orders

Introduction

Stop market orders fill as a market order when the asset reaches a specific price. A buy stop market order triggers when the asset price is greater than or equal to the stop price. A sell stop market order triggers when the asset price is less than or equal to the stop price. You can use stop market orders to buy breakouts or to mitigate the risk of large losses. However, if the market gaps past your stop price, the order may fill at a worse price than the stop price you set. There is no guarantee that a stop market order fills at the stop price.

Place Orders

To send a stop market order, call the `StopMarketOrder` method and provide a `Symbol` , quantity, and stop price. You can also provide a tag and order properties to the `StopMarketOrder` method. If you do not have sufficient capital for the order, it is rejected.

```
StopMarketOrder(symbol, quantity, stopPrice, tag, orderProperties);
```

To buy an asset with a stop market order, set the stop price above the current ask price.

```
// When Bitcoin trades up to $60,000, buy 1 Bitcoin
StopMarketOrder("BTCUSD", 1, 60000);
```

To sell an asset with a stop market order, set the stop price below the current bid price.

```
// When Bitcoin trades down to $43,000, sell 1 Bitcoin
StopMarketOrder("BTCUSD", -1, 43000);
```

Monitor Order Fills

Stop market orders fill as a market order when the security price passes the stop price. To monitor the fills of your order, save a reference to the order ticket .

```
C#
// When XLK trades down to $130, sell 10 shares
var ticket = StopMarketOrder("XLK", -10, 130);
Debug($"Quantity filled: {ticket.QuantityFilled}; Fill price: {ticket.AverageFillPrice}");
```

For more information about how LEAN models order fills in backtests, see Trade Fills .

Update Orders

You can update the quantity, stop price, and tag of stop market orders until the order fills or the brokerage prevents modifications. To update an order, pass an `UpdateOrderFields` object to the `Update` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Update` method returns an OrderResponse to signal the success or failure of the update request.

```
C#
// Create a new order and save the order ticket
var ticket = StopMarketOrder("SPY", -100, 415, tag: "original tag");

// Update the order
var response = ticket.Update(new UpdateOrderFields() {
  Quantity = -80,
  StopPrice = 400,
  Tag = "new tag"
});

// Check if the update was successful
if (response.IsSuccess) {
     Debug("Order updated successfully");
}
```

To update individual fields of an order, call any of the following methods:

- `UpdateQuantity`
- `UpdateStopPrice`
- `UpdateTag`

```
var quantityResponse = ticket.UpdateQuantity(quantity, tag);

var stopResponse = ticket.UpdateStopPrice(stopPrice, tag);

var tagResponse = ticket.UpdateTag(tag);
```

When you update an order, LEAN creates an `UpdateOrderRequest` object, which have the following attributes:

To get a list of `UpdateOrderRequest` objects for an order, call the `UpdateRequests` method.

```
var updateRequests = ticket.UpdateRequests();
```

Cancel Orders

To cancel a stop market order, call the `Cancel` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Cancel` method returns an `OrderResponse` object to signal the success or failure of the cancel request.

```
var response = ticket.Cancel("Cancelled trade");
if (response.IsSuccess)
{
    Debug("Order successfully cancelled");
}
```

When you cancel an order, LEAN creates a `CancelOrderRequest` , which have the following attributes:

To get the `CancelOrderRequest` for an order, call the `CancelRequest` method on the order ticket. The method returns `null` if the order hasn't been cancelled.

```
var request = ticket.cancel_order_request();
```

Asynchronous Orders

When you trade a large portfolio of assets, you may want to send orders in batches and not wait for the response of each one. To send asynchronous orders , set the `asynchronous` argument to `true` .

```
StopMarketOrder("SPY", 10, 429.0m, asynchronous: true);
```

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with stop market orders, set the brokerage model to a brokerage that supports them.

```
SetBrokerageModel(BrokerageName.QuantConnectBrokerage);
```

To check if your brokerage has any special requirements for stop market orders, see the Orders section of the brokerage model documentation .

Requirements

Stop market orders can be submitted at any time for all security types.

Examples

The following backtest verifies the `StopMarketOrder` behavior. On even days, the algorithm buys SPY at the current market price and sells when the price drops 1%. On odd days, the algorithm shorts SPY and buys when the price rises 1%. The following table shows the first four trades in the backtest:

Time	Symbol	Price	Quantity	Type	Status	Value	Tag
2021-07-01T13:31:00Z	SPY	429.10	-1	Market	Filled	-429.10	
2021-07-01T13:31:00Z	SPY	433.44	1	Stop Market	Filled	433.44	Stop Price: 433.43
2021-07-02T17:04:00Z	SPY	433.44	1	Market	Filled	433.44	
2021-07-02T17:04:00Z	SPY	429.00	-1	Stop Market	Filled	-429.00	Stop Price: 429.10

On July 1, 2021, the algorithm shorts SPY at 429.10*andthenbuysitbackat*433.44 when the stop price is 433.43.*Thestoppriceis*1 429.10 is not the market price when the algorithm places the first two orders. 429.10*isthefillpriceatthebid*, *butit*'*snotfarfromthemarketpricebecause*429.1 ∗ 1.01 = 433.39.*Thefillpriceofthestopmarketorderis*433.44, which, as expected, is higher than $433.43. The fill model assumes the worst-case scenario between the market price and the stop price. In this case, the worst-case scenario is the maximum of the market price and stop price.

On July 2, 2021, the algorithm buys SPY at 433.44*andsellsitat*429 when the stop price is 429.10.*Thestoppriceis*1433.44 is not the market price when the algorithm places the second two orders. 433.44*isthefillpriceattheask*, *butit*'*snotfarofffromthemarketpricebecause*433.44 ∗ 0.99 = 429.11.*Thefillpriceofthestopmarketorderis*429, which, as expected, is lower than $429.10. The fill model assumes the worst-case scenario between the market price and the stop price. In this case, the worst-case scenario is the minimum of the market price and stop price.

To reproduce these results, backtest the following algorithm:

C#

```
public class StopMarketOrderAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 7);
        SetCash(100000);
        UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;
        _symbol = AddEquity("SPY").Symbol;
    }

    public override void OnData(Slice data)
    {
        if (!Portfolio.Invested)
        {
            var quantity = 1;
            var factor = 0.99;
            if (Time.Day % 2 != 0)
            {
                quantity = -1;
                factor = 1.01;
            }

            MarketOrder(_symbol, quantity);
            StopMarketOrder(_symbol, -quantity, data[_symbol].Close * (decimal)factor);
        }
    }
}
```

Order Types

Stop Limit Orders

Introduction

Stop limit orders are instructions to place a limit order once an asset passes a specific price level. They are similar to stop market orders , but use limit orders instead of market orders to give you more control over the fill price. A buy stop limit order has a stop price above the current market price and a sell stop limit order has a stop price below the current market price. In effect, they are the opposite of limit if touched orders .

Place Orders

To send a stop limit order, call the `StopLimitOrder` method and provide a `Symbol` , quantity, stop price, and limit price. You can also provide a tag and order properties to the `StopLimitOrder` method. If you do not have sufficient capital for the order, it is rejected.

```
C#
StopLimitOrder(symbol, quantity, stopPrice, limitPrice, tag, orderProperties);
```

To buy an asset with a stop limit order that has a marketable limit price, set the stop price above the current market price and set the limit price above the stop price.

```
C#
// When Bitcoin trades up to $59,000, buy 1 Bitcoin with a limit order at $62,000
StopLimitOrder("BTCUSD", 1, 59000, 62000);
```

To buy an asset with a stop limit order that has an unmarketable limit price, set the stop price above the current market price and set the limit price below the stop price.

```
C#
// When Bitcoin trades up to $52,000, buy 1 Bitcoin with a limit order at $41,000
StopLimitOrder("BTCUSD", 1, 52000, 41000);
```

To sell an asset with a stop limit order that has an unmarketable limit price, set the stop price below the current market price and set the limit price above the stop price.

```C#
// When Bitcoin trades down to $49,000, sell 1 Bitcoin with a limit order at $58,000
StopLimitOrder("BTCUSD", -1, 49000, 58000);
```

To sell an asset with a stop limit order that has a marketable limit price, set the stop price below the current market price and set the limit price below the stop price.

```C#
// When Bitcoin trades down to $37,000, sell 1 Bitcoin with a limit order at $34,000
StopLimitOrder("BTCUSD", -1, 37000, 34000);
```

Monitor Order Fills

Stop limit orders fill as limit orders when the security price passes the stop price. To monitor the fills of your order, save a reference to the order ticket .

```
C#
// When GLD trades up to $200, buy 10 shares with a limit order at $205
var ticket = StopLimitOrder("GLD", 10, 200, 205);
Debug($"Quantity filled: {ticket.QuantityFilled}; Fill price: {ticket.AverageFillPrice}");
```

For more information about how LEAN models order fills in backtests, see Trade Fills .

Update Orders

You can update the quantity, stop price, limit price, and tag of stop limit orders until the order fills or the brokerage prevents modifications. To update an order, pass an `UpdateOrderFields` object to the `Update` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Update` method returns an OrderResponse to signal the success or failure of the update request.

```
C#
// Create a new order and save the order ticket
var ticket = StopLimitOrder("SPY", -10, 400, 390, tag: "original tag");

// Update the order
var response = ticket.Update(new UpdateOrderFields() {
  Quantity = -15,
  StopPrice = 415,
  LimitPrice = 395,
  Tag = "new tag"
});

// Check if the update was successful
if (response.IsSuccess) {
     Debug("Order updated successfully");
}
```

To update individual fields of an order, call any of the following methods:

- `UpdateLimitPrice`
- `UpdateQuantity`

- `UpdateStopPrice`
- `UpdateTag`

```
C#
var limitResponse = ticket.UpdateLimitPrice(limitPrice, tag);
var quantityResponse = ticket.UpdateQuantity(quantity, tag);
var stopResponse = ticket.UpdateStopPrice(stopPrice, tag);
var tagResponse = ticket.UpdateTag(tag);
```

When you update an order, LEAN creates an `UpdateOrderRequest` object, which have the following attributes:

To get a list of `UpdateOrderRequest` objects for an order, call the `UpdateRequests` method.

```
C#
var updateRequests = ticket.UpdateRequests();
```

Asynchronous Orders

When you trade a large portfolio of assets, you may want to send orders in batches and not wait for the response of each one. To send asynchronous orders , set the `asynchronous` argument to `true` .

```
C#
StopLimitOrder("SPY", -10, 428.95m, 429.00m, asynchronous: true);;
```

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with stop limit orders, set the brokerage model to a brokerage that supports them.

```
C#
SetBrokerageModel(BrokerageName.QuantConnectBrokerage);
```

To check if your brokerage has any special requirements for stop limit orders, see the Orders section of the brokerage model documentation .

Requirements

Stop limit orders can be submitted at any time for all security types.

If your algorithm subscribes to extended market hours, they can be filled outside regular trading hours.

Examples

The following backtest verifies the `StopLimitOrder` behavior. The following table shows the trades in the backtest:

Submitted Time	Filled Time	Symbol	Stop Price	Limit Price	Filled Price	Quantity	Type	Status	Value	Tag
2021-07-01T09:31:00Z	2021-07-01T09:37:00Z	SPY	428.95	429.00	429.00	-10	Stop Limit	Filled	-4290.00	Stop Price: ¤428.95 Limit Price: ¤429.00
2021-07-02T09:31:00Z	2021-07-02T10:12:00Z	SPY	431.80	431.75	431.75	10	Stop Limit	Filled	4317.50	Stop Price: ¤431.80 Limit Price: ¤431.75

On July 1, 2021 at 9:31 AM Eastern Time (ET), the algorithm places a stop limit order to sell SPY with a stop price of 428.95*andalimitpriceof*429. At the time of the order submission, the ask high price is429.16*andtheaskclosepriceis*429.11. The order fills at 9:37 AM ET at a price of $429. The fill model triggers the stop for buy orders when the ask high price is greater than the stop price, fills the order when the ask close price is less than the limit price, and sets the fill price to the maximum of the ask high price and the limit price.

On July 2, 2021 at 9:31 AM ET, the algorithm places a stop limit order to buy SPY with a stop price of 431.80*andalimitpriceof* 431.75. At the time of the order submissions, the bid low price is431.49*andthebidclosepriceis*431.54. The order fills at 10:12 AM ET at a price of $431.75. The fill model triggers the stop for sell orders when bid low price is less than the stop price, fills the order when the ask close price is less than the limit price, and sets the fill price to the minimum of the ask high price and the limit price.

C#

```
public class StopLimitOrderAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 4);
        SetCash(100000);
        UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;
        _symbol = AddEquity("SPY").Symbol;
    }

    public override void OnData(Slice data)
    {
        if (Time.Day == 1 && Time.Hour == 9 && Time.Minute == 31)
        {
            StopLimitOrder(_symbol, -10, 428.95m, 429.00m);
        }
        else if (Time.Day == 2 && Time.Hour == 9 && Time.Minute == 31)
        {
            StopLimitOrder(_symbol, 10, 431.80m, 431.75m);
        }
    }
}
```

Order Types

Trailing Stop Orders

Introduction

Trailing stop orders are stop market orders with a dynamic stop price. A sell trailing stop starts with the stop price at a fixed amount below the market price. As the market price rises, the stop price rises by the trail amount, but if the stock price falls, the stop loss price doesn't change. A buy trailing stop order starts with the stop price at a fixed amount above the market price. As the market price falls, the stop price falls by the trail amount, but if the stock price rises, the stop loss price doesn't change.

Trailing stop orders fill as a market order when the asset reaches the stop price. You can use trailing stop orders to buy breakouts or to mitigate the risk of large losses while protecting your gains. However, if the market gaps past your stop price, the order may fill at a worse price than the stop price you set. There's no guarantee that a trailing stop order fills at the stop price.

Place Orders

To send a trailing stop order, call the `TrailingStopOrder` method and provide a `Symbol` , quantity, trailing amount, and the trailing type. You can also provide a tag and order properties to the `TrailingStopOrder` method. If you do not have sufficient capital for the order, it is rejected.

```C#
TrailingStopOrder(symbol, quantity, trailingAmount, trailingAsPercentage, tag, orderProperties);
```

If you set the `trailingAsPercentage` parameter to `false` , the `trailingAmount` is denominated in the asset's quote currency .

Buy With a Currency-Based Trailing Stop

To buy an asset with a currency-based trailing stop order, pass the trailing amount and disable the `trailingAsPercentage` parameter.

```C#
// Buy 1 share of SPY through a trailing stop order that starts $30 above the current price
TrailingStopOrder("SPY", 1, 30m, false);
```

Buy With a Percent-Based Trailing Stop

To buy an asset with a percent-based trailing stop order, pass the trailing percent and enable the `trailingAsPercentage` parameter.

```C#
// Buy 1 share of SPY through a trailing stop order that starts 7% above the current price
TrailingStopOrder("SPY", 1, 0.07m, true);
```

Sell With a Currency-Based Trailing Stop

To sell an asset with a currency-based trailing stop order, pass the trailing amount and disable the `trailingAsPercentage` parameter.

```C#
// Sell 1 share of SPY through a trailing stop order that starts $20 below the current price
TrailingStopOrder("SPY", -1, 20m, false)
```

Sell With a Percent-Based Trailing Stop

To sell an asset with a percent-based trailing stop order, pass the trailing percent and enable the `trailingAsPercentage` parameter.

```C#
// Sell 1 share of SPY through a trailing stop order that starts 5% above the current price
TrailingStopOrder("SPY", -1, 0.05m, true);
```

Custom Starting Prices

In the preceding examples, LEAN calculates the stop price based on the current security price. If you want the order to start with a stop price closer to the security price than the `trailingAmount`, pass a `stopPrice` parameter to the `TrailingStopOrder` method.

```
C#

TrailingStopOrder(symbol, quantity, stopPrice, trailingAmount, trailingAsPercentage, tag, orderProperties);
```

Monitor Order Fills

Trailing stop orders fill as a market order when the security price passes the stop price. To monitor the fills of your order, save a reference to the order ticket.

```
C#

// When XLK trades down to less than 5% of the current price, sell 10 shares, and update
var ticket = TrailingStopOrder("XLK", -10, 0.05, trailingAsPercentage: true);
Debug($"Quantity filled: {ticket.QuantityFilled}; Fill price: {ticket.AverageFillPrice}");
```

For more information about how LEAN models order fills in backtests, see Trade Fills.

Update Orders

The security's fill model automatically updates the stop price of trailing stop orders as the security's price moves away from the stop price. You can update the quantity, stop price, and tag of trailing stop orders until the order fills or the brokerage prevents modifications. To update an order, pass an `UpdateOrderFields` object to the `Update` method on the `OrderTicket`. If you don't have the order ticket, get it from the transaction manager. The `Update` method returns an OrderResponse to signal the success or failure of the update request.

```
C#

// Create a new order and save the order ticket
var ticket = TrailingStopOrder("SPY", -100, 415, 10, trailingAsPercentage: false, tag: "original tag");

// Update the order
var response = ticket.Update(new UpdateOrderFields() {
  Quantity = -80,
  StopPrice = 400,
  Tag = "new tag"
});

// Check if the update was successful
if (response.IsSuccess) {
     Debug("Order updated successfully");
}
```

To update individual fields of an order, call any of the following methods:

- `UpdateQuantity`
- `UpdateStopPrice`
- `UpdateTag`

```
C#

var quantityResponse = ticket.UpdateQuantity(quantity, tag);

var stopResponse = ticket.UpdateStopPrice(stopPrice, tag);

var tagResponse = ticket.UpdateTag(tag);
```

When you update an order, LEAN creates an `UpdateOrderRequest` object, which have the following attributes:

To get a list of `UpdateOrderRequest` objects for an order, call the `UpdateRequests` method.

```
C#
var updateRequests = ticket.UpdateRequests();
```

Cancel Orders

To cancel a trailing stop order, call the `Cancel` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Cancel` method returns an `OrderResponse` object to signal the success or failure of the cancel request.

```
C#
var response = ticket.Cancel("Cancelled trade");
if (response.IsSuccess)
{
    Debug("Order successfully cancelled");
}
```

When you cancel an order, LEAN creates a `CancelOrderRequest` , which have the following attributes:

To get the `CancelOrderRequest` for an order, call the `CancelRequest` method on the order ticket. The method returns `null` if the order hasn't been cancelled.

```
C#
var request = ticket.cancel_order_request();
```

Asynchronous Orders

When you trade a large portfolio of assets, you may want to send orders in batches and not wait for the response of each one. To send asynchronous orders , set the `asynchronous` argument to `true` .

```
C#
TrailingStopOrder("SPY", -100, 0.01m, trailingAsPercentage: true, asynchronous: true);;
```

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with trailing stop orders, set the brokerage model to a brokerage that supports them.

```
C#
SetBrokerageModel(BrokerageName.QuantConnectBrokerage);
```

To check if your brokerage has any special requirements for trailing stop orders, see the Orders section of the brokerage model documentation .

If your brokerage doesn't support them, you can implement a workaround with stop market orders . For an example, see the Buy and Hold with a Trailing Stop Bootcamp lesson.

Requirements

Trailing stop orders can be submitted at any time for all security types.

Examples

The following backtest verifies the `TrailingStopOrder` behavior. On even days, the algorithm buys SPY at the current market price and sells when the price drops 1%. On odd days, the algorithm shorts SPY and buys when the price rises 1%. The following table shows the first four trades in the backtest:

Time	Symbol	Price	Quantity	Type	Status	Value	Tag
2021-07-01T13:31:00Z	SPY	429.10	-1	Market	Filled	-429.10	
2021-07-01T13:31:00Z	SPY	433.16	1	Trailing Stop	Filled	433.16	Stop Price: 433.431400 Trailing Amount: 1.00%
2021-07-02T16:53:00Z	SPY	433.16	1	Market	Filled	433.16	
2021-07-02T16:53:00Z	SPY	429.00	-1	Trailing Stop	Filled	-429.00	Stop Price: 428.823450 Trailing Amount: 1.00%

On July 1, 2021, the algorithm shorts SPY at 429.10*andthenbuysitbackat*433.16 when the stop price is 433.16.*Thestoppriceis*0.95 429.10 is not the market price when the algorithm places the first two orders. The market price is 429.14,*andtheinitialstoppriceis*429.14 ∗ 1.01 = 433.4314.*Themarketpricedropsto*428.86 at 9:40, and the stop price is set to 428.86 * 1.01 = 433.1486. The fill price of the stop market order is 433.16,*which*,*asexpected*,*ishigherthan*433.15. The fill model assumes the worst-case scenario between the market price and the stop price. In this case, the worst-case scenario is the maximum of the market price and stop price.

On July 2, 2021, the algorithm buys SPY at 433.16*andthensellsitat*429 when the stop price is 429.00.*Thestoppriceis*0.96433.16 is not the market price when the algorithm places the second two orders. The market price is 433.155,*andtheinitialstoppriceis*433.155 ∗ 0.99 = 428.82345.*Themarketpricesincreasedto*434.76 on July 7 at 3:35 PM, and the stop price is set to 434.76 * 0.99 = 430.4124. The fill price of the stop market order is 429,*which*,*asexpected*,*islowerthan*430.41. The fill model assumes the worst-case scenario between the market price and the stop price. In this case, the worst-case scenario is the minimum of the market price and stop price in a gap down opening.

To reproduce these results, backtest the following algorithm:

C#

```
public class TrailingStopOrderAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    const decimal _trailingAmount = 0.01m;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 9);
        SetCash(100000);
        _symbol = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;
    }

    public override void OnData(Slice data)
    {
        if (!Portfolio.Invested)
        {
            var quantity = Time.Day % 2 == 0 ? 1 : -1;
            MarketOrder(_symbol, quantity);
            TrailingStopOrder(_symbol, -quantity, _trailingAmount, true);
        }
    }
}
```

Order Types

Market On Open Orders

Introduction

Market on open (MOO) orders fill at the official opening auction price for a security. Market that operate 24/7 don't support MOO orders.

Place Orders

To send a MOO order, call the `MarketOnOpenOrder` method and provide a `Symbol` and quantity. Submit the order at least two minutes before the market opens to be included in the opening auction. If you do not have sufficient capital for the order, it's rejected.

```
C#
// Buy 100 shares of IBM at the market open
MarketOnOpenOrder("IBM", 100);

// Sell 100 shares of IBM at the market open
MarketOnOpenOrder("IBM", -100);
```

You can provide a tag and order properties to the `MarketOnOpenOrder` method.

```
C#
MarketOnOpenOrder(symbol, quantity, tag: tag, orderProperties: orderProperties);
```

Monitor Order Fills

MOO orders fill at the opening auction. If the auction is heavily skewed to one side of the market, your MOO order may not fill. To monitor the fills of your order, save a reference to the order ticket .

```
C#
// Buy 10 shares of SPY at the market open
var ticket = MarketOnOpenOrder("SPY", 10);
Debug($"Quantity filled: {ticket.QuantityFilled}; Fill price: {ticket.AverageFillPrice}");
```

You won't know the fill price of the order until after the market opens. If the asset price moves before the market open to where you can't afford the quantity of the order, the brokerage rejects your order. To increase the probability of a successful trade, leave a sufficient buying power buffer to handle daily price gaps.

For more information about how LEAN models order fills in backtests, see Trade Fills .

Update Orders

You can update the quantity and tag of MOO orders until the order fills or the brokerage prevents modifications. To update an order, pass an `UpdateOrderFields` object to the `Update` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Update` method returns an OrderResponse to signal the success or failure of the update request.

```
// Create a new order and save the order ticket
var ticket = MarketOnOpenOrder("AAPL", 100, tag:"original tag");

// Update the order
var response = ticket.Update(new UpdateOrderFields()
{
    Quantity = 75,
    Tag = "new tag"
});

// Check if the update was successful
if (response.IsSuccess)
{
    Debug("Order updated successfully");
}
```

To update individual fields of an order, call any of the following methods:

- `UpdateQuantity`
- `UpdateTag`

```
var quantityResponse = ticket.UpdateQuantity(quantity, tag);

var tagResponse = ticket.UpdateTag(tag);
```

When you update an order, LEAN creates an `UpdateOrderRequest` object, which have the following attributes:

To get a list of `UpdateOrderRequest` objects for an order, call the `UpdateRequests` method.

```
var updateRequests = ticket.UpdateRequests();
```

Cancel Orders

To cancel a MOO order, call the `Cancel` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Cancel` method returns an `OrderResponse` object to signal the success or failure of the cancel request.

```
var response = ticket.Cancel("Cancelled trade");
if (response.IsSuccess)
{
    Debug("Order successfully cancelled");
}
```

When you cancel an order, LEAN creates a `CancelOrderRequest` , which have the following attributes:

To get the `CancelOrderRequest` for an order, call the `CancelRequest` method on the order ticket. The method returns `null` if the order hasn't been cancelled.

```
var request = ticket.cancel_order_request();
```

Asynchronous Orders

When you trade a large portfolio of assets, you may want to send orders in batches and not wait for the response of each one.

To send asynchronous orders , set the `asynchronous` argument to `true` .

```
C#
MarketOnOpenOrder("SPY", 10, asynchronous: true);
```

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with MOO orders, set the brokerage model to a brokerage that supports them.

```
C#
SetBrokerageModel(BrokerageName.QuantConnectBrokerage);
```

To check if your brokerage has any special requirements for MOO orders, see the Orders section of the brokerage model documentation .

Requirements

You can submit MOO orders at any time for Equity, Equity Options, and Index Options.

Markets that operate 24/7 don't support MOO orders. The Forex market doesn't operate during the weekend. However, MOO orders are invalid if you place them after this market closes for the weekend.

MOO orders don't support the `GoodTilDate` time in force . If you submit a MOO order with the `GoodTilDate` time in force, LEAN automatically adjusts the time in force to be `GoodTilCanceled` .

Examples

The following backtest verifies the `MarketOnOpenOrder` behavior. The following table shows the first trade in the backtest:

Submitted Time	Filled Time	Symbol	Price	Quantity	Type	Status	Value	Tag
2021-07-01T10:31:00Z	2021-07-02T09:31:00Z	SPY	431.67	10	Market On Open	Filled	4316.70	

On July 1, 2021 at 10:31 AM Eastern Time (ET), the algorithm places a market on open order to buy SPY. The fill model fills the order on July 2, 2021 at 9:31 AM at a price of $431.67, which is the official opening auction price for July 2, 2021.

To reproduce these results, backtest the following algorithm:

C#

```
public class MarketOnOpenOrderAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 4);
        SetCash(100000);
        UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;
        _symbol = AddEquity("SPY").Symbol;
    }

    public override void OnData(Slice data)
    {
        if (Time.Day == 1 && Time.Hour == 10 && Time.Minute == 31)
        {
            MarketOnOpenOrder(_symbol, 10);
        }
    }
}
```

Order Types

Market On Close Orders

Introduction

Market on close (MOC) orders fill at the official closing auction price for a security. Markets that operate 24/7 don't support MOC orders.

Place Orders

To send a MOC order, call the `MarketOnCloseOrder` method with a `Symbol` and quantity. If you don't have sufficient capital for the order, it is rejected.

```
C#
// Buy 100 shares of AAPL at the market open
MarketOnCloseOrder("AAPL", 100);

// Sell 100 shares of AAPL at the market open
MarketOnCloseOrder("AAPL", -100);
```

You can provide a tag and order properties to the `MarketOnCloseOrder` method.

```
C#
MarketOnCloseOrder(symbol, quantity, tag: tag, orderProperties: orderProperties);
```

By default, you must place MOC orders at least 15.5 minutes before the close, but some exchanges let you submit them closer to the market closing time. To adjust the buffer period that's required, set the `MarketOnCloseOrder.SubmissionTimeBuffer` property.

```
C#
Orders.MarketOnCloseOrder.SubmissionTimeBuffer = TimeSpan.FromMinutes(10);
```

You can also place MOC orders after the market close.

Monitor Order Fills

MOC orders fill at the closing auction. If the auction is heavily skewed to one side of the market, your MOC order may not fill. To monitor the fills of your order, save a reference to the order ticket .

```
C#
// Buy 10 shares of SPY at the market close
var ticket = MarketOnCloseOrder("SPY", 10);
Debug($"Quantity filled: {ticket.QuantityFilled}; Fill price: {ticket.AverageFillPrice}");
```

You won't know the fill price of the order until after the market closes. If the asset price moves before the market close to where you can't afford the quantity of the order, the brokerage rejects your order. To increase the probability of a successful trade, leave a sufficient buying power buffer to handle the price movements before the close.

For more information about how LEAN models order fills in backtests, see Trade Fills .

Update Orders

You can update the quantity and tag of MOC orders until the order fills or the brokerage prevents modifications. To update an order, pass an `UpdateOrderFields` object to the `Update` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Update` method returns an OrderResponse to signal the success or failure of the update request.

```C#
// Create a new order and save the order ticket
var ticket = MarketOnCloseOrder("SLV", 25, tag: "original tag");

// Update the order
var response = ticket.Update(new UpdateOrderFields()
{
    Quantity = 50,
    Tag = "new tag"
});

// Check if the update was successful
if (response.IsSuccess)
{
    Debug("Order updated successfully");
}
```

To update individual fields of an order, call any of the following methods:

- `UpdateQuantity`
- `UpdateTag`

```C#
var quantityResponse = ticket.UpdateQuantity(quantity, tag);

var tagResponse = ticket.UpdateTag(tag);
```

When you update an order, LEAN creates an `UpdateOrderRequest` object, which have the following attributes:

To get a list of `UpdateOrderRequest` objects for an order, call the `UpdateRequests` method.

```C#
var updateRequests = ticket.UpdateRequests();
```

Cancel Orders

To cancel a MOC order, call the `Cancel` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Cancel` method returns an `OrderResponse` object to signal the success or failure of the cancel request.

```C#
var response = ticket.Cancel("Cancelled trade");
if (response.IsSuccess)
{
    Debug("Order successfully cancelled");
}
```

When you cancel an order, LEAN creates a `CancelOrderRequest` , which have the following attributes:

To get the `CancelOrderRequest` for an order, call the `CancelRequest` method on the order ticket. The method returns `null` if the order hasn't been cancelled.

```
var request = ticket.cancel_order_request();
```

Asynchronous Orders

When you trade a large portfolio of assets, you may want to send orders in batches and not wait for the response of each one. To send asynchronous orders , set the `asynchronous` argument to `true` .

```
MarketOnCloseOrder("SPY", 10, asynchronous: true);
```

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with MOC orders, set the brokerage model to a brokerage that supports them.

```
SetBrokerageModel(BrokerageName.QuantConnectBrokerage);
```

To check if your brokerage has any special requirements for MOC orders, see the Orders section of the brokerage model documentation .

Requirements

By default, you must place MOC orders at least 15.5 minutes before the close, but some exchanges let you submit them closer to the market closing time. To adjust the buffer period that's required, set the `MarketOnCloseOrder.SubmissionTimeBuffer` property.

```
Orders.MarketOnCloseOrder.SubmissionTimeBuffer = TimeSpan.FromMinutes(10);
```

Markets that operate 24/7 don't support MOC orders. The Forex market doesn't operate during the weekend.

MOC orders don't support the `GoodTilDate` time in force . If you submit a MOC order with the `GoodTilDate` time in force, LEAN automatically adjusts the time in force to be `GoodTilCanceled` .

Examples

The following backtest verifies the `MarketOnCloseOrder` behavior. The following table shows the first trade in the backtest:

Submitted Time	Filled Time	Symbol	Price	Quantity	Type	Status	Value	Tag
2021-07-01T10:31:00Z	2021-07-01T16:00:00Z	SPY	430.43	10	Market On Close	Filled	4304.30	

On July 1, 2021 at 10:31 AM Eastern Time (ET), the algorithm places a market on open order to buy SPY. The fill model fills the order on July 1, 2021 at 4:00 PM ET at a price of $430.43, which is the official closing auction price for July 1, 2021.

To reproduce these results, backtest the following algorithm:

C#

```
public class MarketOnCloseOrderAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 4);
        SetCash(100000);
        UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;
        _symbol = AddEquity("SPY").Symbol;
    }

    public override void OnData(Slice data)
    {
        if (Time.Day == 1 && Time.Hour == 10 && Time.Minute == 31)
        {
            MarketOnCloseOrder(_symbol, 10);
        }
    }
}
```

Order Types

Combo Market Orders

Introduction

Combo market orders are individual orders that contain market orders for muliple securities. Combo market orders currently only work for trading Option contracts.

Place Orders

To send a combo market order, create multiple `Leg` objects to represent the legs of the combo order, then call the `ComboMarketOrder` method. At least one leg must have a positive quantity and at least one leg must have a negative quantity. The legs must each target a unique contract. If you don't have sufficient capital for the order, it's rejected. By default, combo market orders are synchronous and fill immediately.

```C#
foreach (var kvp in slice.OptionChains)
{
    // Select contracts
    var contracts = kvp.Value.Contracts.Values.ToList();
    if (contracts.Count < 2)
    {
        return;
    }

    // Create order legs
    var legs = new List<Leg>()
    {
        Leg.Create(contracts[0].Symbol, 1),
        Leg.Create(contracts[1].Symbol, -1)
    };

    // Place order
    ComboMarketOrder(legs, 1);
}
```

The quantity of the legs sets the ratio of the leg orders while the quantity argument of the `ComboMarketOrder` method sets the combo order size and acts as a global multiplier. In the preceding example, if we set the global multiplier to two, then the algorithm buys two units of the first contract and sells two units of the second contract. The quantity argument of the `ComboMarketOrder` method also sets the order direction of the combo order, which affects how the fill model fills the order.

You can also provide a tag and order properties to the `ComboMarketOrder` method.

```C#
ComboMarketOrder(legs, quantity, tag: tag, orderProperties: orderProperties);
```

Monitor Order Fills

If the brokerage has sufficient liquidity in their order book, combo market orders fill immediately. Otherwise, you get partial fills. To monitor the fills of your order, save a reference to the order tickets .

```
var tickets = ComboMarketOrder(legs, 1);
foreach (var ticket in tickets)
{
    Debug($"Symbol: {ticket.Symbol}; Quantity filled: {ticket.QuantityFilled}; Fill price:
{ticket.AverageFillPrice}");
}
```

For more information about how LEAN models order fills in backtests, see Trade Fills .

Cancel Orders

To cancel a combo market order, call the `Cancel` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Cancel` method returns an `OrderResponse` object to signal the success or failure of the cancel request.

```
var response = ticket.Cancel("Cancelled trade");
if (response.IsSuccess)
{
    Debug("Order successfully cancelled");
}
```

When you cancel an order, LEAN creates a `CancelOrderRequest` , which have the following attributes:

To get the `CancelOrderRequest` for an order, call the `CancelRequest` method on the order ticket. The method returns `null` if the order hasn't been cancelled.

```
var request = ticket.cancel_order_request();
```

Asynchronous Orders

When you trade a large portfolio of assets, you may want to send orders in batches and not wait for the response of each one. To send asynchronous orders , set the `asynchronous` argument to `true` .

```
ComboMarketOrder(legs, quantity, asynchronous: true);;
```

Synchronous Timeouts

Combo market orders are synchronous by default, so your algorithm waits for the order to fill before moving to the next line of code. If your order takes longer than five seconds to fill, your algorithm continues executing even if the trade isn't filled. To adjust the timeout period, set the `Transactions.MarketOrderFillTimeout` property.

```
// Adjust the market fill-timeout to 30 seconds.
Transactions.MarketOrderFillTimeout = TimeSpan.FromSeconds(30);
```

Combo market orders may take a few minutes to fill for illiquid assets such as out-of-the-money Options.

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with combo market orders, set the brokerage model to a brokerage that supports them.

```
SetBrokerageModel(BrokerageName.QuantConnectBrokerage);
```

To check if your brokerage has any special requirements for combo market orders, see the Orders section of the brokerage model documentation .

Requirements

You can submit combo market orders during regular trading hours for all asset classes. For Futures and Future Options, you can place combo market orders during regular and extended market hours. For all other asset classes, if you place combo market orders at/after the last minute of regular market hours, LEAN converts the combo market orders into market-on-open orders as long as the asset class supports them . To view the trading hours of each asset class, follow these steps:

1. Open the Asset Classes documentation.
2. Click an asset class.
3. Click Market Hours .

Examples

The following backtest verifies the `ComboMarketOrder` behavior. The algorithm buys one contract and sells one contract at the same time. The following table shows the two trades in the backtest:

Time	Symbol	Price	Quantity	Type	Status	Value	Tag
2015-12-24T09:31:00Z	GOOG 16011SC00745000	16.90	1	Buy	Filled	16.90	
2015-12-24T09:31:00Z	GOOG 160115C00747500	14.20	-1	Sell	Filled	-14.20	

On December 24, 2015, the algorithm buys GOOG 16011SC00745000 at $16.90 and sells GOOG 160115C00747500 at $14.20. The fill model fills the buy order at the ask close price and fills the sell order at the bid close price.

To reproduce these results, backtest the following algorithm:

C#

```
public class ComboMarketOrderAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 12, 24);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        var option = AddOption("GOOG");
        option.SetFilter(minStrike: -2, maxStrike: 2, minExpiry: TimeSpan.FromDays(0), maxExpiry:
TimeSpan.FromDays(180));

    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested)
        {
            return;
        }
        foreach (var kvp in slice.OptionChains)
        {
            var contracts = kvp.Value.Contracts.Values.ToList();
            if (contracts.Count < 2)
            {
                return;
            }

            var legs = new List<Leg>()
            {
                Leg.Create(contracts[0].Symbol, 1),
                Leg.Create(contracts[1].Symbol, -1)
            };

            var tickets = ComboMarketOrder(legs, 1);
        }
    }
}
```

Order Types

Combo Limit Orders

Introduction

Combo limit orders are individual orders that contain limit orders for muliple securities. Combo limit orders are different from combo leg limit orders because you must set the limit price of all the leg orders to be the same with combo limit orders. With combo leg limit orders, you can create the order without forcing each leg to have the same limit price. Combo limit orders currently only work for trading Option contracts and their underlying Equities.

Place Orders

To send a combo limit order, create multiple `Leg` objects to represent the legs of the combo order, then call the `ComboLimitOrder` method. At least one leg must have a positive quantity and a tleast one leg must have a negative quantity. The legs must each target a unique contract, but don't set the `OrderPrice` property for any of the legs. If you don't have sufficient capital for the order, it's rejected.

```C#
foreach (var kvp in slice.OptionChains)
{
    // Select contracts
    var contracts = kvp.Value.Contracts.Values.ToList();
    if (contracts.Count < 2)
    {
        return;
    }

    // Create order legs
    var legs = new List&lgt;Leg>()
    {
        Leg.Create(contracts[0].Symbol, 1),
        Leg.Create(contracts[1].Symbol, -1)
    };

    // Calculate limit price
    var limitPrice = Math.Round(legs.Select(leg => Securities[leg.Symbol].Close).Sum() * 0.95m, 2);

    // Place order
    ComboLimitOrder(legs, quantity: 1, limitPrice: limitPrice);
}
```

The quantity of the legs sets the ratio of the leg orders while the quantity argument of the `ComboLimitOrder` method sets the combo order size and acts as a global multiplier. In the preceding example, if we set the global multiplier to two, then the algorithm buys two units of the first contract and sells two units of the second contract. The quantity also sets the order direction of the combo limit order, which affects how the fill model fills the order.

You can also provide a tag and order properties to the `ComboLimitOrder` method.

```C#
ComboLimitOrder(legs, quantity, limitPrice, tag: tag, orderProperties: orderProperties);
```

Monitor Order Fills

Combo limit orders fill all the legs at the same time. Each leg can fill when the sum of the security prices of the legs pass the limit price of the combo order. To monitor the fills of your order, save a reference to the order tickets .

```
var tickets = ComboLimitOrder(legs, 1, limitPrice);
foreach (var ticket in tickets)
{
    Debug($"Symbol: {ticket.Symbol}; Quantity filled: {ticket.QuantityFilled}; Fill price:
{ticket.AverageFillPrice}");
}
```

For more information about how LEAN models order fills in backtests, see Trade Fills .

Update Orders

You can update the quantity, limit price, and tag of the limit orders in each leg until the combo order fills or the brokerage prevents modifications. To update an order, pass an `UpdateOrderFields` object to the `Update` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . To update the limit price of the combo order, you only need to update the limit price of one of the leg orders. The `Update` method returns an OrderResponse to signal the success or failure of the update request.

```
// Create a new order and save the order ticket
var tickets = ComboLimitOrder(legs, quantity: 1, limitPrice: limitPrice);

// Update the leg orders
foreach (var ticket in tickets)
{
    var response = ticket.Update(new UpdateOrderFields()
    {
        Quantity = 2 * Math.Sign(ticket.Quantity),
        LimitPrice = ticket.Get(OrderField.LimitPrice) + 0.01m,
        Tag = $"Update #{ticket.UpdateRequests.Count + 1}"
    });

    // Check if the update was successful
    if (response.IsSuccess)
    {
        Debug($"Order updated successfully for {ticket.Symbol}");
    }
}
```

To update individual fields of an order, call any of the following methods:

- `UpdateLimitPrice`
- `UpdateQuantity`
- `UpdateTag`

```
var limitResponse = ticket.UpdateLimitPrice(limitPrice, tag);

var quantityResponse = ticket.UpdateQuantity(quantity, tag);

var tagResponse = ticket.UpdateTag(tag);
```

When you update an order, LEAN creates an `UpdateOrderRequest` object, which have the following attributes:

To get a list of `UpdateOrderRequest` objects for an order, call the `UpdateRequests` method.

```
var updateRequests = ticket.UpdateRequests();
```

Cancel Orders

To cancel a combo limit order, call the `Cancel` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Cancel` method returns an `OrderResponse` object to signal the success or failure of the cancel request.

```
C#
var response = ticket.Cancel("Cancelled trade");
if (response.IsSuccess)
{
    Debug("Order successfully cancelled");
}
```

When you cancel an order, LEAN creates a `CancelOrderRequest` , which have the following attributes:

To get the `CancelOrderRequest` for an order, call the `CancelRequest` method on the order ticket. The method returns `null` if the order hasn't been cancelled.

```
C#
var request = ticket.cancel_order_request();
```

Asynchronous Orders

When you trade a large portfolio of assets, you may want to send orders in batches and not wait for the response of each one. To send asynchronous orders , set the `asynchronous` argument to `true` .

```
C#
ComboLimitOrder(legs, 1, limitPrice, asynchronous: true);
```

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with combo limit orders, set the brokerage model to a brokerage that supports them.

```
C#
SetBrokerageModel(BrokerageName.QuantConnectBrokerage);
```

To check if your brokerage has any special requirements for combo limit orders, see the Orders section of the brokerage model documentation .

Requirements

Combo limit orders can be submitted at any time for all security types.

If your algorithm subscribes to extended market hours, they can be filled outside regular trading hours.

Examples

The following backtest verifies the `ComboLimitOrder` behavior. The algorithm buys one contract and sells one contract at the same time. The following table shows the two trades in the backtest:

Time	Symbol	Price	Quantity	Type	Status	Value	Tag
2015-12-24T09:31:00Z	GOOG 16011SC00745000	16.50	1	Buy	Filled	16.50	
2015-12-24T09:31:00Z	GOOG 160115C00747500	14.60	-1	Sell	Filled	-14.60	

On December 24, 2015 at 9:31 AM Eastern Time (ET), the algorithm places a combo limit order to buy one GOOG 16011SC00745000 contract and sell one GOOG 160115C00747500 contracts. The limit price is 75% of the contract spread, which equals 2.02.*Thecomboorderdoesn*′*tfillimmediatelybecausethecontractspread*2.40 > 2.02.*By*9:36*AM*,*thespreaddropsto*1.90, which is below the limit price, so the fill model fills the combo limit order.

To reproduce these results, backtest the following algorithm:

C#

```
public class ComboMarketOrderAlgorithm : QCAlgorithm
{
    private List<OrderTicket> _tickets = new();
    private decimal _limitPrice;
    public override void Initialize()
    {
        SetStartDate(2024, 12, 24);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("GOOG");
        option.SetFilter(minStrike: -2, maxStrike: 2, minExpiry: TimeSpan.FromDays(0), maxExpiry:
TimeSpan.FromDays(180));
    }

    public override void OnData(Slice slice)
    {
        if (_tickets.IsNullOrEmpty())
        {
            foreach (var kvp in slice.OptionChains)
            {
                // Select contracts
                var contracts = kvp.Value.Where(contract => contract.Right == OptionRight.Call)
                    .GroupBy(x => x.Expiry)
                    .OrderBy(grouping => grouping.Key)
                    .First()
                    .OrderBy(x => x.Strike)
                    .ToList();

                if (contracts.Count < 2)
                {
                    return;
                }

                // Create order legs
                var legs = new List<Leg>()
                {
                    Leg.Create(contracts[0].Symbol, 1),
                    Leg.Create(contracts[1].Symbol, -1)
                };

                // Calculate limit price
                _limitPrice = Math.Round((slice.QuoteBars[contracts[0].Symbol].Ask.Low -
slice.QuoteBars[contracts[1].Symbol].Bid.Low) * 0.75m, 2);

                // Place order
                _tickets = ComboLimitOrder(legs, 1, _limitPrice);
            }
        }
        else
        {
            if (Time.Hour == 9 && Time.Minute == 37)
            {
                Quit();
                return;
            }

            // Log the limit price and aggregate contract price
            var price = 0.0m;
            foreach (var ticket in _tickets)
            {
                var quoteBar = slice.QuoteBars[ticket.Symbol];
                price += (ticket.Quantity > 0 ? quoteBar.Ask : quoteBar.Bid).Low * Math.Sign(ticket.Quantity);
            }
            Log($"{Time}. Limit price: {_limitPrice}; Aggregate price: {Math.Round(price, 2)}; Ready to fill: {price
< _limitPrice}");
        }
    }
    public override void OnOrderEvent(OrderEvent orderEvent)
    {
        if (orderEvent.Status == OrderStatus.Filled)
        {
            Log($"{Time} -- Order {orderEvent.OrderId} filled at {orderEvent.FillPrice}");
        }
    }
}
```

Order Types

Combo Leg Limit Orders

Introduction

Combo leg limit orders are individual orders that contain limit orders for muliple securities. Combo leg limit orders are different from combo limit orders because you can create combo leg limit orders without forcing each leg to have the same limit price. Combo leg limit orders currently only work for trading Option contracts.

Place Orders

To send a combo leg limit order, create multiple `Leg` objects to represent the legs of the combo order, then call the `ComboLegLimitOrder` method. The legs must each target a unique contract. At least one leg must have a positive quantity and at least one leg must have a negative quantity. If you don't have sufficient capital for the order, it's rejected.

```C#
foreach (var kvp in slice.OptionChains)
{
    // Select contracts
    var contracts = kvp.Value.Contracts.Values.ToList();
    if (contracts.Count < 2)
    {
        return;
    }

    // Create order legs
    var legs = new List<Leg>()
    {
        Leg.Create(contracts[0].Symbol, 1, contracts[0].LastPrice * 0.98m),
        Leg.Create(contracts[1].Symbol, -1, contracts[1].LastPrice * 1.02m)
    };

    // Place order
    ComboLegLimitOrder(legs, 1);
}
```

The quantity of the legs sets the ratio of the leg orders while the quantity argument of the `ComboLegLimitOrder` method sets the combo order size and acts as a global multiplier. In the preceding example, if we set the global multiplier to two, then the algorithm buys two units of the first contract and sells two units of the second contract. The quantity also sets the order direction of the combo limit order, which affects how the fill model fills the order.

You can also provide a tag and order properties to the `ComboLegLimitOrder` method.

```C#
ComboLegLimitOrder(legs, quantity, tag: tag, orderProperties: orderProperties);
```

Monitor Order Fills

Combo leg limit orders fill all the legs at the same time. Each leg can fill when the security price passes the limit price of the leg. To monitor the fills of your order, save a reference to the order tickets .

```
var tickets = ComboLegLimitOrder(legs, 1);
foreach (var ticket in tickets)
{
    Debug($"Symbol: {ticket.Symbol}; Quantity filled: {ticket.QuantityFilled}; Fill price:
{ticket.AverageFillPrice}");
}
```

For more information about how LEAN models order fills in backtests, see Trade Fills .

Update Orders

You can update the quantity, limit price, and tag of the leg limit orders until the combo order fills or the brokerage prevents modifications. To update an order, pass an `UpdateOrderFields` object to the `Update` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Update` method returns an OrderResponse to signal the success or failure of the update request.

```
// Create a new order and save the order ticket
var tickets = ComboLegLimitOrder(legs, 1);

// Update the leg orders
foreach (var ticket in tickets)
{
    var direction = Math.Sign(ticket.Quantity);
    var response = ticket.Update(new UpdateOrderFields()
    {
        Quantity = 2 * direction,
        LimitPrice = ticket.Get(OrderField.LimitPrice) + 0.01m * direction,
        Tag = $"Update #{ticket.UpdateRequests.Count + 1}"
    });

    // Check if the update was successful
    if (response.IsSuccess)
    {
        Debug($"Order updated successfully for {ticket.Symbol}");
    }
}
```

To update individual fields of an order, call any of the following methods:

- `UpdateLimitPrice`
- `UpdateQuantity`
- `UpdateTag`

```
var limitResponse = ticket.UpdateLimitPrice(limitPrice, tag);

var quantityResponse = ticket.UpdateQuantity(quantity, tag);

var tagResponse = ticket.UpdateTag(tag);
```

When you update an order, LEAN creates an `UpdateOrderRequest` object, which have the following attributes:

To get a list of `UpdateOrderRequest` objects for an order, call the `UpdateRequests` method.

```
var updateRequests = ticket.UpdateRequests();
```

Cancel Orders

To cancel a combo leg limit order, call the `Cancel` method on the `OrderTicket` . If you don't have the order ticket, get it from the transaction manager . The `Cancel` method returns an `OrderResponse` object to signal the success or failure of the cancel request.

```
C#
var response = ticket.Cancel("Cancelled trade");
if (response.IsSuccess)
{
    Debug("Order successfully cancelled");
}
```

When you cancel an order, LEAN creates a `CancelOrderRequest` , which have the following attributes:

To get the `CancelOrderRequest` for an order, call the `CancelRequest` method on the order ticket. The method returns `null` if the order hasn't been cancelled.

```
C#
var request = ticket.cancel_order_request();
```

Asynchronous Orders

When you trade a large portfolio of assets, you may want to send orders in batches and not wait for the response of each one. To send asynchronous orders , set the `asynchronous` argument to `true` .

```
C#
ComboLegLimitOrder(legs, 1, asynchronous: true);
```

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with combo leg limit orders, set the brokerage model to a brokerage that supports them.

```
C#
SetBrokerageModel(BrokerageName.QuantConnectBrokerage);
```

To check if your brokerage has any special requirements for combo leg limit orders, see the Orders section of the brokerage model documentation .

Requirements

Combo leg limit orders can be submitted at any time for all security types.

If your algorithm subscribes to extended market hours, they can be filled outside regular trading hours.

Examples

The following backtest verifies the `ComboLegLimitOrder` behavior. The algorithm buys one contract and sells one contract at the same time. The following table shows the two trades in the backtest:

Time	Symbol	Price	Quantity	Type	Status	Value	Tag
2015-12-24T09:31:00Z	GOOG 16011SC00745000	16.10	2	Buy	Filled	32.20	Update #72
2015-12-24T09:31:00Z	GOOG 160115C00747500	14.11515	-2	Sell	Filled	-28.2303	Update #72

On December 24, 2015 at 9:31 AM Eastern Time (ET), the algorithm places a combo leg limit order to buy one GOOG 16011SC00745000 contract and sell two GOOG 160115C00747500 contracts. The limit price of both orders is 99.9% of the respective contract price, which is 16.2837*forGOOG*16011*SC*00745000*and*14.83515 for GOOG 160115C00747500. The combo order doesn't fill immediately, so the algorithm updates the leg orders at each time step. During the first update, the algorithm sets the quantity of the GOOG 160115C00747500 leg to -2. During each update, the limit price moves 0.01*closertothemarket*. *Thatis*, *thelimitpriceofGOOG*16011*SC*00745000*increasesby*0.01 and the limit price of GOOG 160115C00747500 decreases by $0.01. After the 72nd update, the ask low price is below the limit price of the leg to buy GOOG 16011SC00745000 and the bid high price is above the limit price of the leg to sell GOOG 160115C00747500, so the fill model fills the combo leg limit order at 10:44 AM ET.

To reproduce these results, backtest the following algorithm:

C#

```
public class ComboLegLimitOrderAlgorithm : QCAlgorithm
{
    private List<OrderTicket> _tickets = new();

    public override void Initialize()
    {
        SetStartDate(2024, 12, 24);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("GOOG");
        option.SetFilter(minStrike: -2, maxStrike: 2, minExpiry: TimeSpan.FromDays(0), maxExpiry:
TimeSpan.FromDays(180));
    }

    public override void OnData(Slice slice)
    {
        if (_tickets.IsNullOrEmpty())
        {
            foreach (var kvp in slice.OptionChains)
            {
                // Select contracts
                var contracts = kvp.Value.Where(contract => contract.Right == OptionRight.Call)
                    .GroupBy(x => x.Expiry)
                    .OrderBy(grouping => grouping.Key)
                    .First()
                    .OrderBy(x => x.Strike)
                    .ToList();

                if (contracts.Count < 2)
                {
                    return;
                }

                // Create order legs
                var legs = new List<Leg>()
                {
                    Leg.Create(contracts[0].Symbol, 1, slice[contracts[0].Symbol].Close * 0.999m),
                    Leg.Create(contracts[1].Symbol, -2, slice[contracts[1].Symbol].Close * 0.999m)
                };

                // Place order
                _tickets = ComboLegLimitOrder(legs, 1);
            }
        }
        else
        {
            if (Time.Hour == 10 && Time.Minute == 45)
```

```
            {
                Quit();
                return;
            }

            foreach (var ticket in _tickets)
            {
                var orderDirection = Math.Sign(ticket.Quantity);
                var limitPrice = ticket.Get(OrderField.LimitPrice);

                // Log the limit prices and contract prices
                var quoteBar = slice.QuoteBars[ticket.Symbol];
                var currentPrice = orderDirection == 1 ? quoteBar.Ask.Low : quoteBar.Bid.High;
                var readyToFill = orderDirection == 1 ? currentPrice < limitPrice : currentPrice > limitPrice;
                Log($"{Time} - {ticket.Symbol}. Current price: {currentPrice}; Limit price: {Math.Round(limitPrice,
2)}; Order direction: {orderDirection}; Ready to fill: {readyToFill}");

                // Update the leg orders
                var response = ticket.Update(new UpdateOrderFields()
                {
                    Quantity = 2 * orderDirection,
                    LimitPrice = limitPrice + 0.01m * orderDirection,
                    Tag = $"Update #{ticket.UpdateRequests.Count + 1}"
                });
            }
        }
    }

    public override void OnOrderEvent(OrderEvent orderEvent)
    {
        if (orderEvent.Status == OrderStatus.Filled)
        {
            Log($"{Time} -- Order {orderEvent.OrderId} filled at {orderEvent.FillPrice}");
        }
    }
}
```

Order Types

Option Exercise Orders

Introduction

If you buy an Option contract, you can exercise your right to buy or sell the underlying asset at the strike price. However, you don't need to exercise it. You can sell the Option or let it expire worthless if it's out of the money. If you hold a long position in an Option that expires in the money, LEAN automatically exercises it at the expiration date. If you sell an Option contract and the buyer exercises their right to buy or sell the underlying asset, you are assigned the Option and must trade with the buyer at the strike price.

Place Orders

You can exercise American-style Option contracts anytime before they expire. Depending on your brokerage and the delivery method, you may be able to exercise European-style Option contracts on their expiration date. To exercise an Option, call the `ExerciseOption` method with the Option contract `Symbol` and a quantity. If you do not have sufficient capital for the order, it's rejected. By default, Option exercise orders are synchronous and fill immediately.

```
C#
var ticket = ExerciseOption(contractSymbol, quantity);
```

You can provide a tag and order properties to the `ExerciseOption` method.

```
C#
ExerciseOption(symbol, quantity, tag: tag, orderProperties: orderProperties);
```

Monitor Order Fills

To monitor the fills of your order, save a reference to the order ticket .

```
C#
var ticket = ExerciseOption(contractSymbol, quantity);
Debug($"Quantity filled: {ticket.QuantityFilled}; Fill price: {ticket.AverageFillPrice}");
```

For more information about how LEAN models Option exercise orders in backtests, see the Exercise Option model .

Option Assignments

If you sell an Option in a backtest, LEAN can simulate an Option exercise order on behalf of the buyer. By default, LEAN scans your portfolio every hour. It considers exercising American-style Options if they are within 4 days of their expiration and it considers exercising European-style Options on their day of expiration. If you have sold an Option that's 5% in-the-money and the Option exercise order is profitable after the cost of fees, LEAN exercises the Option. For more information about how we simulate Option assignments, see the Assignment reality model.

Asynchronous Orders

When you trade a large portfolio of assets, you may want to send orders in batches and not wait for the response of each one. To send asynchronous orders , set the `asynchronous` argument to `true` .

```
var ticket = ExerciseOption(contractSymbol, quantity, asynchronous: true);
```

Synchronous Timeouts

Option exercise orders are synchronous by default, so your algorithm waits for the order to fill before moving to the next line of code. If your order takes longer than five seconds to fill, your algorithm continues executing even if the trade isn't filled. To adjust the timeout period, set the `Transactions.MarketOrderFillTimeout` property.

```
// Adjust the market fill-timeout to 30 seconds.
Transactions.MarketOrderFillTimeout = TimeSpan.FromSeconds(30);
```

Brokerage Support

Each brokerage has a set of assets and order types they support. To avoid issues with Option exercise orders, set the brokerage model to a brokerage that supports them.

```
SetBrokerageModel(BrokerageName.QuantConnectBrokerage);
```

To check if your brokerage has any special requirements for Option exercise orders, see the Orders section of the brokerage model documentation .

Requirements

Option exercise orders can only be submitted for option contracts with a long position. European-style options cannot be exercised before their expiration date.

Examples

The following backtest verifies the `ExerciseOption` behavior. The following table shows the first three trades in the backtest:

Time	Symbol	Price	Quantity	Type	Status	Value	Tag
2021-07-01T09:31:00Z	SPY 221216C00085000	345.30	1	Buy Market	Filled	34530.00	
2021-07-01T09:31:00Z	SPY 221216C00085000	0.00	-1	Sell Option Exercise	Filled	0.00	Automatic Exercise
2021-07-01T09:31:00Z	SPY	85.00	100	Sell Option Exercise	Filled	8500.00	Option Exercise

The algorithm first brought a deep ITM option based on the set option selection conditions by 345.30, *thenexerciseitactively*, *so*100*sharesofSPY*(*inwhich*1*contractrepresents*)*wasboughtatitsstrikepriceat*85.00, with the option discarded from the portfolio.

To reproduce these results, backtest the following algorithm:

C#

```
public class OptionExerciseOrderAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private decimal _strike;
    private Symbol _contract;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 4);
        SetCash(100000);
        Settings.SeedInitialPrices = true;
        AddSecurityInitializer(CustomSecurityInitializer);

        var spy = AddEquity("SPY", Resolution.Minute);
        _spy = spy.Symbol;
    }

    private void CustomSecurityInitializer(Security security)
    {
        security.SetDataNormalizationMode(DataNormalizationMode.Raw);
    }

    public override void OnData(Slice data)
    {
        if (!Portfolio.Invested)
        {
            var contracts = OptionChainProvider.GetOptionContractList(_spy, Time);
            contracts = from contract in contracts
                        where contract.ID.OptionRight == OptionRight.Call
                        orderby contract.ID.StrikePrice ascending
                        select contract;
            if (contracts.Count() == 0) return;

            _strike = contracts.First().ID.StrikePrice;

            contracts = from contract in contracts
                        where contract.ID.StrikePrice == _strike
                        orderby contract.ID.Date ascending
                        select contract;

            _contract = contracts.First();
            AddOptionContract(_contract, Resolution.Minute);

            Buy(_contract, 1);
        }

        if (Portfolio[_contract].Invested && _strike < Securities[_spy].Price)
        {
            ExerciseOption(_contract, 1);
            Quit();
        }
    }
}
```

Order Types

Other Order Types

Introduction

We are often asked to support other order types like one cancels the other, trailing stop, and multi-leg orders. Currently, LEAN doesn't support these order types, but we will add them over time. Part of the difficulty of implementing them is the incomplete brokerage support.

One Cancels the Other Orders

One cancels the other (OCO) orders are a set of orders that when one fills, it cancels the rest of the orders in the set. An example is to set a take-profit and a stop-loss order right after you enter a position. In this example, when either the take-profit or stop-loss order fills, you cancel the other order. OCO orders usually create an upper and lower bound on the exit price of a trade.

When you place OCO orders, their price levels are usually relative to the fill price of an entry trade. If your entry trade is a synchronous market order, you can immediately get the fill price from the order ticket. If your entry trade doesn't execute immediately, you can get the fill price in the OnOrderEvents event handler. Once you have the entry fill price, you can calculate the price levels for the OCO orders.

```C#
// Get the fill price from the order ticket of a sync market order
_market = MarketOrder("SPY", 1);
var fillPrice = _market.AverageFillPrice;

// Get the fill price from the OnOrderEvent event handler
public override void OnOrderEvent(OrderEvent orderEvent)
{
    if (orderEvent.Status == OrderStatus.Filled && orderEvent.Ticket.OrderType == OrderType.Market)
    {
        var fillPrice = orderEvent.FillPrice;
    }
}
```

After you have the target price levels, to implement the OCO orders, you can place active orders or track the security price to simulate the orders.

Place Active Orders

To place active orders for the OCO orders, use a combination of limit orders and stop limit orders . Place these orders so that their price levels that are far enough apart from each other. If their price levels are too close, several of the orders can fill in a single time step. When one of the orders fills, in the `OnOrderEvent` event handler, cancel the other orders in the OCO order set.

```C#
private OrderTicket _stopLoss;
private OrderTicket _takeProfit;

public override void OnOrderEvent(OrderEvent orderEvent)
{
    if (orderEvent.Status != OrderStatus.Filled) return;

    switch (orderEvent.Ticket.OrderType)
    {
        case OrderType.Market:
            _stopLoss = StopMarketOrder(orderEvent.Symbol, -orderEvent.FillQuantity, orderEvent.FillPrice*0.95m);
            _takeProfit = LimitOrder(orderEvent.Symbol, -orderEvent.FillQuantity, orderEvent.FillPrice*1.10m);
            return;
        case OrderType.StopMarket:
            _takeProfit?.Cancel();
            return;
        case OrderType.Limit:
            _stopLoss?.Cancel();
            return;
    }
}
```

Simulate Orders

To simulate OCO orders, track the asset price in the `OnData` method and place market or limit orders when asset price reaches the take-profit or stop-loss level. The benefit of manually simulating the OCO orders is that both of the orders can't fill in the same time step.

```C#
decimal _entryPrice;

public override void OnData(Slice slice)
{
    if (!Portfolio.Invested)
    {
        var ticket = MarketOrder("SPY", 1);
        _entryPrice = ticket.AverageFillPrice;
    }

    if (!slice.Bars.ContainsKey("SPY")) return;

    if (slice.Bars["SPY"].Price >= _entryPrice * 1.10m)
    {
        Liquidate(symbol: "SPY", -1, tag: "take profit");
    }
    else if (slice.Bars["SPY"].Price <= _entryPrice * 0.95m)
    {
        Liquidate(symbol: "SPY", -1, tag: "stop loss");
    }
}
```

Multi-Leg Orders

Multi-leg orders are orders that contain multiple sub-orders. Examples of multi-leg orders include Option strategies like spreads , straddles , and strangles . You can manually implement other types of multi-leg orders with the built-in order types.

Trading and Orders

Position Sizing

Introduction

LEAN provides several methods to help you set specific portfolio weights for assets. These methods account for lot size and pre-calculated order fees.

Single Asset Targets

The `SetHoldings` method calculates the number of asset units to purchase based on the portfolio weight you provide and then submits market orders . This method provides a quick way to set up a portfolio with a set of weights for assets. If you already have holdings, you may want to liquidate the existing holdings first to free up buying power.

```C#
// Allocate 50% of portfolio value to IBM
SetHoldings("IBM", 0.5);

// Allocate 50% of portfolio value to IBM, but liquidate other holdings first
SetHoldings("IBM", 0.5, liquidateExistingHoldings: true);

// Provide a tag and order properties to the SetHoldings method
SetHoldings(symbol, weight, liquidateExistingHoldings, tag, orderProperties);
```

If the percentage you provide translates to an order quantity of 0 or to a small quantity that doesn't exceed the `MinimumOrderMarginPortfolioPercentage` algorithm setting , then the `SetHoldings` method doesn't place an order and doesn't log anything.

Multiple Asset Targets

When you trade a weighted basket of assets, sometimes you must intelligently scale down existing positions before increasing allocations to other assets. If you call the `SetHoldings` method with a list of `PortfolioTarget` objects, LEAN sorts the orders based on your position delta and then places the orders that reduce your position size in an asset before it places orders that increase your position size in an asset. When you call the `SetHoldings` method with a list of `PortfolioTarget` objects, the decimal you pass to the `PortfolioTarget` constructor represents the portfolio weight. In this situation, don't use the `PortfolioTarget.Percent` method.

```C#
// Purchase a portfolio of targets, processing orders intelligently.
var targets = new List<PortfolioTarget>() { new ("SPY", 0.8m), new ("IBM", 0.2m) };
SetHoldings(targets);
```

If you want the targets you pass to define the target portfolio state, enable the `liquidateExistingHoldings` argument.

```C#
SetHoldings(targets, liquidateExistingHoldings: true);
```

The method also accepts tag and order properties arguments.

```
SetHoldings(targets, liquidateExistingHoldings, tag, orderProperties);
```

If the percentage you provide translates to an order quantity of 0 or to a small quantity that doesn't exceed the `MinimumOrderMarginPortfolioPercentage` algorithm setting , then the `SetHoldings` method doesn't place an order and doesn't log anything.

Asynchronous Orders

When you trade a large portfolio of assets, you may want to send orders in batches and not wait for the response of each one. To send asynchronous orders , set the `asynchronous` argument to `true` .

```
SetHoldings("IBM", 0.5, asynchronous: true);
```

When you set multiple asset targets, you likely want to send asynchronous orders.

```
var targets = new List<PortfolioTarget>() { new("SPY", 0.8m), new("IBM", 0.2m) };
SetHoldings(targets, asynchronous: true);
```

Calculate Order Quantities

The `SetHoldings` method uses market order to reach the target portfolio weight you provide. If you want to use a different order type, you need to specify the quantity to trade. To calculate the number of units to trade based on a portfolio weight, call the `CalculateOrderQuantity` method. The method calculates the quantity based on the current price of the asset and adjusts it for the fee model of the security. The target weight you provide is an unleveraged value. For instance, if you have 2x leverage and request a 100% weight, the method calculates the quantity that uses half of your available margin.

```
// Calculate the fee-adjusted quantity of shares with given buying power
var quantity = CalculateOrderQuantity("IBM", 0.4);
```

Buying Power Buffer

If you place a market on open order near the market close, the market can gap overnight to a worse open price. If the gap is large enough against your trade direction, you may not have sufficient buying power at the open to fill your trade. To ensure a high probability of order fills through market gaps and discontinuities, the `SetHoldings` method assumes a 0.25% cash buffer. If LEAN rejects your orders due to buying power, widen the cash buffer through the algorithm `Settings` property.

```
// Set the cash buffer to 5%
Settings.FreePortfolioValuePercentage = 0.05m;

// Set the cash buffer to $10,000
Settings.FreePortfolioValue = 10000m;
```

If you use `FreePortfolioValuePercentage` , you must set it in the `Initialize` or `PostInitialize` event handler. If you use `FreePortfolioValue` , you must set it after the `PostInitialize` event handler.

Examples

The following examples demonstrate some common practices for position sizing.

Example 1: Covered Call

```C#
public class PositionSizingAlgorithm : QCAlgorithm
{
    private Symbol _spy;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(10_000_000);

        // Seed the price to ensure securities have information for margin calculations.
        Settings.SeedInitialPrices = true;

        // Request SPY data for trading. We use raw price data to compare the strike price fairly for ATM contract
selection.
        _spy = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;

        // Set a scheduled event to rebalance the covered call positions.
        Schedule.On(
            DateRules.WeekStart(_spy),
            TimeRules.AfterMarketOpen(_spy, 1),
            Rebalance
        );
    }

    private void Rebalance()
    {
        var optionChain = OptionChain(_spy);
        // Filter for the calls within 1-week expiry.
        var expiry = optionChain.Where(x => x.Expiry <= Time.AddDays(6))
            .Max(x => x.Expiry);
        // Select the ATM call.
        var atmCall = optionChain.Where(x => x.Right == OptionRight.Call && x.Expiry == expiry)
            .OrderBy(x => Math.Abs(x.Strike - Securities[_spy].Price))
            .First();
        var atmCallSymbol = AddOptionContract(atmCall).Symbol;

        // Calculate the order size for the covered call.
        var quantity = (int)Math.Floor(CalculateOrderQuantity(_spy, 0.5) / 100m);

        // Covered call involves 100 SPY shares and a -1 ATM call contract.
        if (quantity > 0)
        {
            ComboMarketOrder(
            [
                Leg.Create(_spy, 100),
                Leg.Create(atmCallSymbol, -1)
            ], quantity,
            asynchronous: true);
        }
    }
}
```

Trading and Orders

Liquidating Positions

Introduction

The `Liquidate` method lets you liquidate individual assets or your entire portfolio. The method creates market orders to close positions and returns the tickets of the liquidation orders. If you have pending open orders for the security when you call `Liquidate` , LEAN tries to cancel them. The `Liquidate` method works for all asset classes, except Crypto. To liquidate Crypto positions, see Crypto Trades .

Liquidate Individual Positions

To liquidate your holdings in an individual security, call the `Liquidate` method and provide a ticker or `Symbol` .

```C#
// Liquidate all IBM in your portfolio
var orderTickets = Liquidate("IBM");
```

You can pass an order tag and properties to the `Liquidate` method.

```C#
var orderProperties = new OrderProperties() { TimeInForce = TimeInForce.Day };
var orderTickets = Liquidate("AAPL", tag: "Liquidated", orderProperties: orderProperties);
```

Liquidate All Positions

To liquidate all of the positions in your portfolio, call the `Liquidate` method without any ticker of `Symbol` arguments.

```C#
// Liquidate your entire portfolio
var orderTickets = Liquidate();
```

You can pass an order tag and properties to the `Liquidate` method.

```C#
var orderProperties = new OrderProperties() { TimeInForce = TimeInForce.Day };
var orderTickets = Liquidate(tag: "Liquidated", orderProperties: orderProperties);
```

Asynchronous Liquidations

When you trade a large portfolio of assets, you may want to send orders in batches and not wait for the response of each one. To send asynchronous orders , set the `asynchronous` argument to `true` .

```C#
var orderTickets = Liquidate(asynchronous: true, orderProperties: orderProperties);
```

Enable and Disable Liquidations

By default, the `Liquidate` method is functional. To enable and disable it, set the `LiquidateEnabled` algorithm setting .

```
// Disable liquidations
Settings.LiquidateEnabled = false;

// Enable liquidations
Settings.LiquidateEnabled = true;
```

Market Closed Considerations

If you liquidate your positions when the market is closed, LEAN converts the orders into market on open orders. If your brokerage doesn't support market on open orders , the order is invalid.

Examples

The following examples demonstrate some common practices for liquidating positions.

Example 1: Liquidate Universe Removal

The following algorithm holds equal positions for the top 10 liquid universe to follow their hype while updating weekly. When a stock is removed from the universe, we liquidate the position in the `OnSecuritiesChanged` method.

```
public class LiquidatingPositionsAlgorithm : QCAlgorithm
{
    private Universe _universe;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        Settings.SeedInitialPrices = true;
        SetWarmUp(TimeSpan.FromDays(7), Resolution.Minute);

        // Update the universe weekly to allow time to capitalize on the hype.
        UniverseSettings.Schedule.On(DateRules.WeekStart("SPY"));
        UniverseSettings.Resolution = Resolution.Minute;
        // Filter for the 10 most popular stocks to invest the hype.
        _universe = AddUniverse(Universe.Top(10));

        // Set a schedule event to rebalance weekly.
        Schedule.On(
            DateRules.WeekStart("SPY"),
            TimeRules.AfterMarketOpen("SPY", 1),
            Rebalance
        );
    }

    private void Rebalance()
    {
        // Evenly invest in the universe members to dissipate the capital risk equally.
        var count = _universe.Members.Count;
        var targets = _universe.Members.Select(x => new PortfolioTarget(x.Key, 1m / count)).ToList();
        SetHoldings(targets);
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        // Liquidate the ones leaving the universe since they are not the most popular.
        // Note that the liquidation will be market-on-open orders due to the time of securities removal.
        foreach (var removed in changes.RemovedSecurities)
        {
            if (Securities.TryGetValue(removed.Symbol, out var security) && security.Invested)
            {
                Liquidate(removed.Symbol);
            }
        }
    }
}
```

Example 2: Liquidate Existing Positions

The example below trades the same strategy as the previous one. Still, instead of liquidating the positions when the stocks leave the universe, we liquidate the position using the `liquidateExistingHoldings` argument in the `SetHoldings` . Note that the time of the liquidation will be slightly delayed due to the time of the scheduled event.

```C#
public class LiquidatingPositionsAlgorithm : QCAlgorithm
{
    private Universe _universe;
    private Symbol spy;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        spy = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
        // Update the universe weekly to allow time to capitalize on the hype.
        UniverseSettings.Schedule.On(DateRules.WeekStart(spy));
        // Filter for the 10 most popular stocks to invest the hype.
        _universe = AddUniverse(Universe.Top(10));

        // Set a schedule event to rebalance weekly.
        Schedule.On(
            DateRules.WeekStart(spy),
            TimeRules.AfterMarketOpen(spy, 1),
            Rebalance
        );
    }

    private void Rebalance()
    {
        // Evenly invest in the universe members to dissipate the capital risk equally.
        var activeMembers = _universe.Members
            .Where(kvp => kvp.Value.HasData && kvp.Value.Price > 0)
            .ToList();
        var count = activeMembers.Count;
        var targets = activeMembers.Select(x => new PortfolioTarget(x.Key, 1m / count)).ToList();
        // We can use `liquidateExistingHoldings` arguments to liquidate the ones not in the universe anymore.
        SetHoldings(targets, liquidateExistingHoldings: true);
    }
}
```

Example 3: Liquidate Upon Assignment

The following algorithm sells SPY ATM puts weekly to earn the premium when SPY price is rising. In the case where the price of SPY drops and the put is being assigned, we liquidate the SPY position in the `OnAssignmentOrderEvent` .

C#

```
public class LiquidatingPositionsAlgorithm : QCAlgorithm
{
    private Symbol _spy;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Seed the price with the last known price to ensure the underlying price data is available on initial
option contract filtering.
        Settings.SeedInitialPrices = true;

        // Request SPY data for option selection using its price.
        // Use raw data normalization mode since we need to compare the SPY price with the strike price.
        _spy = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;

        // Set a schedule event to rebalance weekly.
        Schedule.On(
            DateRules.WeekStart(),
            TimeRules.At(9, 31),
            Rebalance
        );
    }

    private void Rebalance()
    {
        // Select the 1-week ATM put SPY option to short.
        var optionChain = OptionChain(_spy);
        var expiry = optionChain.Where(x => x.Expiry < Time.AddDays(6))
            .Max(x => x.Expiry);
        var atmPut = optionChain.Where(x => x.Right == OptionRight.Put && x.Expiry == expiry)
            .OrderBy(x => Math.Abs(x.Strike - Securities[_spy].Price))
            .First();
        var atmPutSymbol = AddOptionContract(atmPut).Symbol;

        // Sell the ATM put option to earn the premium.
        Sell(atmPutSymbol, 2);
    }

    public override void OnAssignmentOrderEvent(OrderEvent assignmentEvent)
    {
        // Liquidate any assigned positions of SPY since we only earn the premium.
        Liquidate(_spy);
    }
}
```

Trading and Orders

Crypto Trades

Introduction

All fiat and Crypto currencies are individual assets. When you buy a pair like BTCUSDT, you trade USDT for BTC. In this case, LEAN removes some USDT from your portfolio cash book and adds some BTC. The virtual pair BTCUSDT represents your position in the trade, but the virtual pair doesn't actually exist. It simply represents an open trade.

Place Trades

When you place Crypto trades, don't use the `CalculateOrderQuantity` or `SetHoldings` methods. Instead, calculate the order quantity based on the currency amounts in your cash book and place manual orders.

The following code snippet demonstrates how to allocate 90% of your portfolio to BTC.

```C#
public override void OnData(Slice data)
{
    SetCryptoHoldings(_symbol, 0.9m);
}

private void SetCryptoHoldings(Symbol symbol, decimal percentage)
{
    var crypto = Securities[symbol] as Crypto;
    var baseCurrency = crypto.BaseCurrency;

    // Calculate the target quantity in the base currency
    var targetQuantity = percentage * (Portfolio.TotalPortfolioValue - Settings.FreePortfolioValue) /
baseCurrency.ConversionRate;
    var quantity = targetQuantity - baseCurrency.Amount;

    // Round down to observe the lot size
    var lotSize = crypto.SymbolProperties.LotSize;
    quantity = Math.Round(quantity / lotSize) * lotSize;

    if (IsValidOrderSize(crypto, quantity))
    {
        MarketOrder(symbol, quantity);
    }
}

// Brokerages have different order size rules
// Binance considered the minimum volume (price x quantity):
private bool IsValidOrderSize(Crypto crypto, decimal quantity)
{
    return Math.Abs(crypto.Price * quantity) > crypto.SymbolProperties.MinimumOrderSize;
}
```

The preceding example doesn't take into account order fees. You can add a 0.1% buffer to accommodate it.

The following example demonstrates how to form an equal-weighted Crypto portfolio and stay within the cash buffer .

```C#
public override void OnData(Slice data)
{
    var percentage = (1m - Settings.FreePortfolioValuePercentage) / _symbols.Count;
    foreach (var symbol in _symbols)
    {
        SetCryptoHoldings(_symbol, percentage);
    }
}
```

You can replace the `Settings.FreePortfolioValuePercentage` for a class variable (e.g. `_cashBuffer`).

When you place Crypto trades, ensure you have a sufficient balance of the base or quote currency before each trade. If you hold multiple assets and you want to put all of your capital into BTCUSDT, you need to first convert all your non-BTC assets into USDT and then purchase BTCUSDT.

By default, the account currency is USD and your starting cash is $100,000. Some Crypto exchanges don't support fiat currencies, which means you can't convert the 100,000 USD to the pair's quote currency . In this case, set your account currency to the quote currency with a positive quantity.

```C#
SetAccountCurrency("USDT", 1000);   // Set the account currency to Tether and its quantity to 1,000 USDT
```

For a full example of placing crypto trades, see the BasicTemplateCryptoAlgorithm .

Liquidate Positions

If you use the `Liquidate` method to liquidate a Crypto position, it only liquidates the quantity of the virtual pair. Since the virtual pair BTCUSDT may not represent all of your BTC holdings, don't use the `Liquidate` method to liquidate Crypto positions. Instead, calculate the order quantity based on the currency amounts in your cash book and place manual orders. The following code snippet demonstrates how to liquidate a BTCUSDT position.

```C#
public override void OnData(Slice data)
{
    LiquidateCrypto(_symbol);
}

private void LiquidateCrypto(Symbol symbol)
{
    var crypto = Securities[symbol] as Crypto;
    var baseCurrency = crypto.BaseCurrency;

    // Avoid negative amount after liquidate
    var quantity = Math.Min(crypto.Holdings.Quantity, baseCurrency.Amount);

    // Round down to observe the lot size
    var lotSize = crypto.SymbolProperties.LotSize;
    quantity = (Math.Round(quantity / lotSize) - 1) * lotSize;

    if (IsValidOrderSize(crypto, quantity))
    {
        MarketOrder(symbol, -quantity);
    }
}
```

The order fees don't respect the lot size. When you try to liquidate a position, the absolute value of the base currency quantity can be less than the lot size and greater than zero. In this case, your algorithm holds a position that you can't liquidate, yet `Portfolio[symbol].Invested` is `false` . For more information about `Portfolio[symbol].Invested` , see Holdings Status . The following code snippet demonstrates how to determine if you can liquidate a position:

```C#
public override void OnData(Slice data)
{
    var crypto = Securities[_symbol];
    if (Math.Abs(crypto.Holdings.Quantity) > crypto.SymbolProperties.LotSize)
    {
        LiquidateCrypto(_symbol);
    }
}
```

Examples

The following examples demonstrate some common practices for crypto holdings.

Example 1: ETH-BTC Proxy Trading

The following algorithm trades trend-following of the ETHBTC crypto pair using a 20-day EMA indicator. To reduce friction (e.g. slippage), we trade the more liquid and popular BTCUSDT and ETHUSDT pair instead. For example, if ETHBTC is above EMA (uptrend), we sell all BTCUSDT holdings and buy ETHUSDT with the same USDT value, going flat on BTC while holding ETH.

C#

```
public class CryptoHoldingsAlgorithm : QCAlgorithm
{
    private Symbol _btcusdt, _ethusdt, _ethbtc;
    private ExponentialMovingAverage _ema;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetAccountCurrency("USDT", 1000000m);

        // We would like to trade the EMA cross between 2 popular cryptos: BTC & ETH,
        // so request ETHBTC data to find trading opportunities.
        _ethbtc = AddCrypto("ETHBTC", Resolution.Daily, market: Market.Coinbase).Symbol;
        // Trade through BTCUSDT & ETHUSDT, though, since stable coin trades have lower friction costs and higher
liquidity.
        var btcusdt = AddCrypto("BTCUSDT", Resolution.Daily, market: Market.Coinbase);
        var ethusdt = AddCrypto("ETHUSDT", Resolution.Daily, market: Market.Coinbase);
        _btcusdt = btcusdt.Symbol;
        _ethusdt = ethusdt.Symbol;

        // Simulate an account with various crypto cash through holdings.
        btcusdt.Holdings.SetHoldings(btcusdt.Price, 5m);
        btcusdt.BaseCurrency.AddAmount(5m);
        ethusdt.Holdings.SetHoldings(ethusdt.Price, 22.5m);
        ethusdt.BaseCurrency.AddAmount(22.5m);

        // Add automatic updating of the EMA indicator for trend trade signal emission.
        _ema = EMA(_ethbtc, 20, Resolution.Daily);
        // Warm up the indicator for its readiness usage immediately.
        WarmUpIndicator(_ethbtc, _ema, Resolution.Daily);
    }

    public override void OnData(Slice slice)
    {
        if (slice.Bars.TryGetValue(_ethbtc, out var bar) && _ema.IsReady &&
            slice.Bars.TryGetValue(_btcusdt, out var btc) && slice.Bars.TryGetValue(_ethusdt, out var eth))
        {
            var ema = _ema.Current.Value;
            var btcHoldings = Portfolio.CashBook["BTC"].Amount;
            var ethHoldings = Portfolio.CashBook["ETH"].Amount;

            // ETHBTC's current price is higher than EMA, suggesting an uptrend.
            if (bar.Close > ema && !Portfolio[_btcusdt].IsShort)
            {
                // To follow the up trend of ETHBTC, sell ALL BTCUSDT and buy ETHUSDT with the same value.
                if (btcHoldings > 0)
                {
                    var btcValue = btcHoldings * btc.Close;
                    MarketOrder(_btcusdt, -btcHoldings);
                    var ethSize = btcValue / eth.Close;
                    MarketOrder(_ethusdt, ethSize);
                }
            }
            // ETHBTC's current price is below the EMA, suggesting a downtrend.
            else if (bar.Close < ema && !Portfolio[_btcusdt].IsLong)
            {
                // To follow the downtrend of ETHBTC, sell ALL ETHUSDT and buy BTCUSDT with the same value.
                if (ethHoldings > 0)
                {
                    var ethValue = ethHoldings * eth.Close;
                    MarketOrder(_ethusdt, -ethHoldings);
                    var btcSize = ethValue / btc.Close;
                    MarketOrder(_btcusdt, btcSize);
                }
            }
        }
    }
}
```

Trading and Orders

Option Strategies

You can implement any of the following Option strategies in an algorithm. Click one to learn more.

Bear Call Spread

Bear Put Spread

Bull Call Spread

Bull Put Spread

Long Call Butterfly

Short Call Butterfly

Long Put Butterfly

Short Put Butterfly

Long Call Calendar Spread

Short Call Calendar Spread

Long Put Calendar Spread

Short Put Calendar Spread

Covered Call

Covered Put

Long Iron Butterfly

Short Iron Butterfly

Long Iron Condor

Short Iron Condor

Naked Call

Naked Put

Protective Call

Protective Put

Protective Collar

Long Straddle

Short Straddle

Long Strangle

Short Strangle

Conversion

Reverse Conversion

Long Box Spread

Short Box Spread

Long Jelly Roll

Short Jelly Roll

Bear Call Ladder

Bear Put Ladder

Bull Call Ladder

Bull Put Ladder

Long Call Backspread

Short Call Backspread

Long Put Backspread

Short Put Backspread

See Also

Equity Options
Future Options
Index Options

Option Strategies

Bear Call Spread

Introduction

Bear call spread , also known as short call spread , consists of selling an ITM call and buying an OTM call. Both calls have the same underlying Equity and the same expiration date. The ITM call serves as a hedge for the OTM call. The bear call spread profits from a drop in underlying asset price.

Implementation

Follow these steps to implement the bear call spread strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(500000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().CallSpread(30, 5));
}
```

   The `CallSpread` filter narrows the universe down to just the two contracts you need to form a bear call spread.

2. In the `OnData` method, select the expiration and strikes of the contracts in the strategy legs.

```
C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select the call Option contracts with the furthest expiry
    var expiry = chain.Max(x => x.Expiry);
    var calls = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Call);
    if (calls.Count() == 0) return;

    // Select the ITM and OTM contract strike prices from the remaining contracts
    var strikes = calls.Select(x => x.Strike).ToList();
    var otmStrike = strikes.Max();
    var itmStrike = strikes.Min();
```

3. In the `OnData` method, select the contracts and place the orders.

   Approach A: Call the `OptionStrategies.BearCallSpread` method with the details of each leg and then pass the result to the `Buy` method.

```C#
var optionStrategy = OptionStrategies.BearCallSpread(_symbol, itmStrike, otmStrike, expiry);
Buy(optionStrategy, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

```C#
var itmCall = calls.Single(x => x.Strike == itmStrike);
var otmCall = calls.Single(x => x.Strike == otmStrike);

var legs = new List<Leg>()
    {
        Leg.Create(itmCall.Symbol, -1),
        Leg.Create(otmCall.Symbol, 1)
    };
ComboMarketOrder(legs, 1);
```

Strategy Payoff

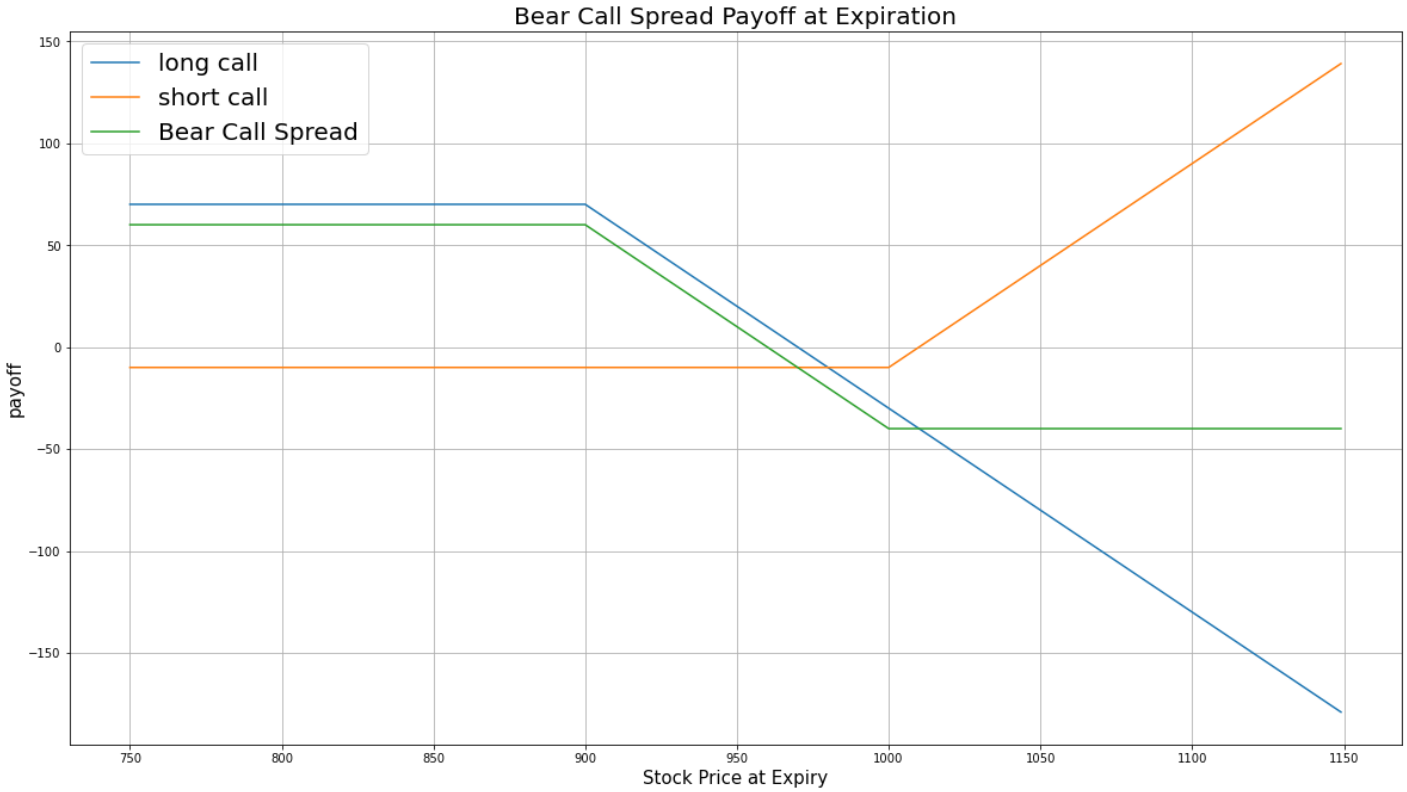
The bear call spread is a limited-reward-limited-risk strategy. The payoff is

$$
\begin{aligned}
C_T^{OTM} &= (S_T - K^{OTM})^+ \\
C_T^{ITM} &= (S_T - K^{ITM})^+ \\
P_T &= (C_T^{OTM} - C_T^{ITM} + C_0^{ITM} - C_0^{OTM}) \times m - fee
\end{aligned}
$$

where

C_T^{OTM}	=	OTM call value at time T
C_T^{ITM}	=	ITM call value at time T
S_T	=	Underlying asset price at time T
K^{OTM}	=	OTM call strike price
K^{ITM}	=	ITM call strike price
P_T	=	Payout total at time T
C_0^{ITM}	=	ITM call value at position opening (debit paid)
C_0^{OTM}	=	OTM call value at position opening (credit received)
m	=	Contract multiplier
T	=	Time of expiration

The following chart shows the payoff at expiration:



The maximum profit is the net credit you receive from opening the trade, $C_0^{ITM} - C_0^{OTM}$. If the price declines, both calls expire worthless.

The maximum loss is $K^{OTM} - K^{ITM} + C_0^{ITM} - C_0^{OTM}$, which occurs when the underlying price is above the strike prices of both call Option contracts.

If the Option is American Option, there is a risk of early assignment on the contract you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	**Price ($)**	**Strike ($)**
OTM call	26.90	1197.50
ITM call	57.80	1125.00
Underlying Equity at expiration	1078.92	-

Therefore, the payoff is

$$
\begin{aligned}
C_T^{OTM} &= (S_T - K^{OTM})^+ \\
&= (1078.92 - 1197.50)^+ \\
&= 0 \\
C_T^{ITM} &= (S_T - K^{ITM})^+ \\
&= (1078.92 - 1125.00)^+ \\
&= 0 \\
P_T &= (C_T^{OTM} - C_T^{ITM} + C_0^{ITM} - C_0^{OTM}) \times m - fee \\
&= (0 - 0 + 57.80 - 26.90) \times 100 - 1.00 \times 2 \\
&= 3088
\end{aligned}
$$

So, the strategy profits $3,088.

The following algorithm implements a bear call spread Option strategy:

```C#
public class BearCallSpreadExampleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(500000);

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().CallSpread(30, 5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested) return;

        // Get the OptionChain
        if (!slice.OptionChains.TryGetValue(_symbol, out var chain)) return;

        // Select the call Option contracts with the furthest expiry
        var expiry = chain.OrderByDescending(x => x.Expiry).First().Expiry;
        var calls = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Call);
        if (calls.Count() == 0) return;

        // Select the ITM and OTM contract
        var orderedCalls = calls.OrderBy(x => x.Strike);
        var itmCall = orderedCalls.First();
        var otmCall = orderedCalls.Last();

        // Place the order
        var optionStrategy = OptionStrategies.BearCallSpread(_symbol, itmCall.Strike, otmCall.Strike, expiry);
        Buy(optionStrategy, 1);
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm
IndexOptionBearCallSpreadAlgorithm.cs C#

Option Strategies

Bear Put Spread

Introduction

Bear put spread , also known as short put spread , consists of buying an ITM put and selling an OTM put. Both puts have the same underlying Equity and the same expiration date. The OTM put serves as a hedge for the ITM put. The bear put spread profits from a decline in underlying asset price.

Implementation

Follow these steps to implement the bear put spread strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(500000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().PutSpread(30, 5));
}
```

   The `PutSpread` filter narrows the universe down to just the two contracts you need to form a bear put spread.

2. In the `OnData` method, select the expiration and strikes of the contracts in the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select the put Option contracts with the furthest expiry
    var expiry = chain.Max(x => x.Expiry);
    var puts = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Put);
    if (puts.Count() == 0) return;

    // Select the ITM and OTM contract strike prices from the remaining contracts
    var strikes = puts.Select(x => x.Strike).ToList();
    var otmStrike = strikes.Min();
    var itmStrike = strikes.Max();
```

3. In the `OnData` method, select the contracts and place the orders.

   Approach A: Call the `OptionStrategies.BearPutSpread` method with the details of each leg and then pass the result to the `Buy` method.

```C#
var optionStrategy = OptionStrategies.BearPutSpread(_symbol, itmStrike, otmStrike, expiry);
Buy(optionStrategy, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

```C#
var itmPut = puts.Single(x => x.Strike == itmStrike);
var otmPut = puts.Single(x => x.Strike == otmStrike);

var legs = new List<Leg>()
    {
        Leg.Create(itmPut.Symbol, 1),
        Leg.Create(otmPut.Symbol, -1)
    };
ComboMarketOrder(legs, 1);
```

Strategy Payoff

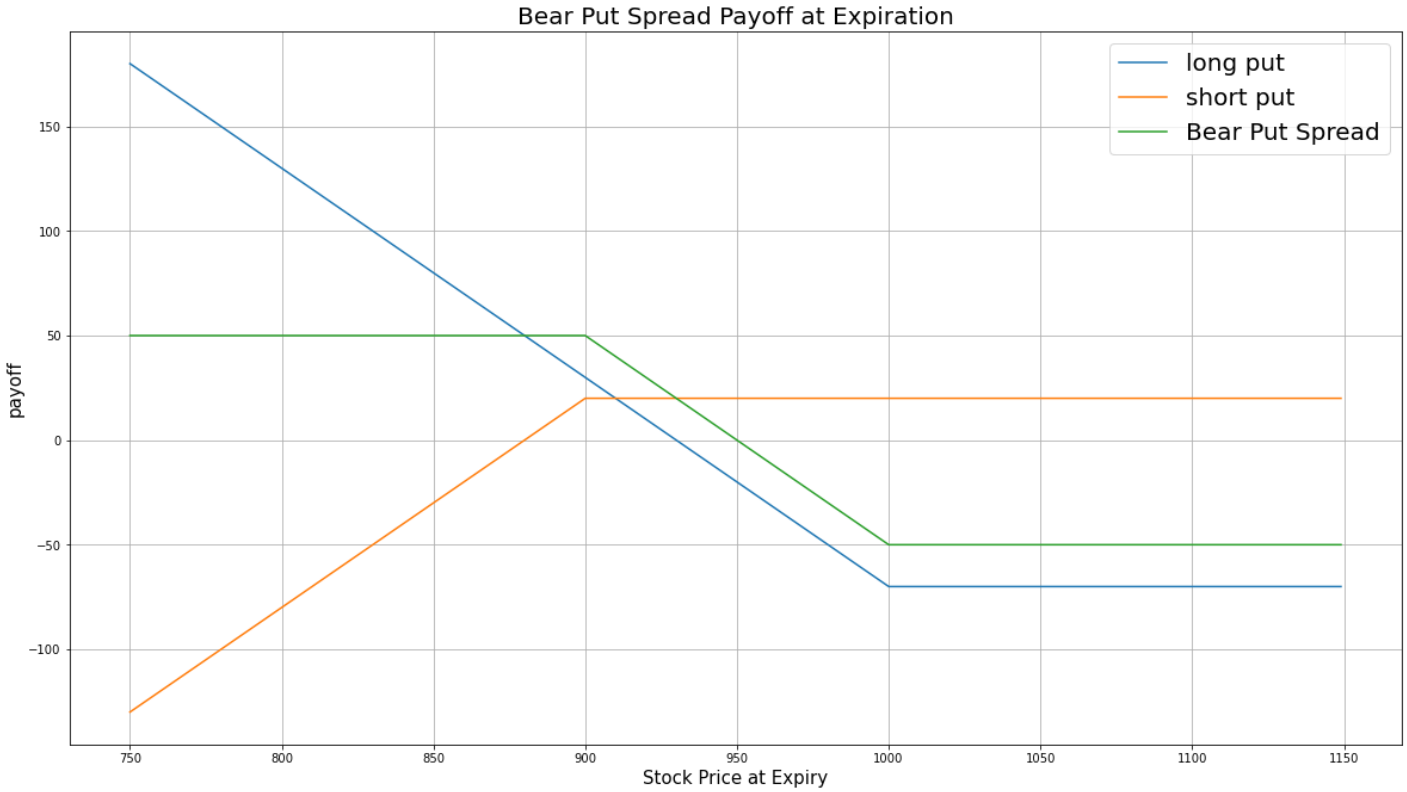
The bear put spread is a limited-reward-limited-risk strategy. The payoff is

$$
\begin{aligned}
P_T^{OTM} &= (K^{OTM} - S_T)^+ \\
P_T^{ITM} &= (K^{ITM} - S_T)^+ \\
P_T &= (P_T^{ITM} - P_T^{OTM} + P_0^{OTM} - P_0^{ITM}) \times m - fee
\end{aligned}
$$

where

P_T^{OTM}	=	OTM put value at time T
P_T^{ITM}	=	ITM put value at time T
S_T	=	Underlying asset price at time T
K^{OTM}	=	OTM put strike price
K^{ITM}	=	ITM put strike price
P_T	=	Payout total at time T
P_0^{ITM}	=	ITM put value at position opening (debit paid)
P_0^{OTM}	=	OTM put value at position opening (credit received)
m	=	Contract multiplier
T	=	Time of expiration

The following chart shows the payoff at expiration:



The maximum profit is $K^{ITM} - K^{OTM} + P_0^{OTM} - P_0^{ITM}$. If the underlying price is below than the strike prices of both put Option contracts, they are worth $(K - S_T)$ at expiration.

The maximum loss is the net debit you paid to open the position, $P_0^{OTM} - P_0^{ITM}$.

If the Option is American Option, there is a risk of early assignment on the contract you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	Price ($)	Strike ($)
OTM put	4.60	767.50
ITM put	40.00	835.00
Underlying Equity at expiration	829.08	-

Therefore, the payoff is

$$
\begin{aligned}
P_T^{OTM} &= (K^{OTM} - S_T)^+ \\
&= (767.50 - 829.08)^+ \\
&= 0 \\
P_T^{ITM} &= (K^{ITM} - S_T)^+ \\
&= (835.00 - 829.08)^+ \\
&= 5.92 \\
P_T &= (P_T^{ITM} - P_T^{OTM} + P_0^{OTM} - P_0^{ITM}) \times m - fee \\
&= (5.92 - 0 + 4.60 - 40.00) \times 100 - 1.00 \times 2 \\
&= -2950
\end{aligned}
$$

So, the strategy loses $2,950.

The following algorithm implements a bear put spread strategy:

```C#
public class BearPutSpreadExampleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(500000);

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().PutSpread(30, 5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested) return;

        // Get the OptionChain
        var chain = slice.OptionChains.get(_symbol, null);
        if (chain.Count() == 0) return;

        // Get the furthest expiry date of the contracts
        var expiry = chain.OrderByDescending(x => x.Expiry).First().Expiry;

        // Select the put Option contracts with the furthest expiry
        var puts = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Put);
        if (puts.Count() == 0) return;

        // Select the ITM and OTM contract strike prices from the remaining contracts
        var putStrikes = puts.Select(x => x.Strike).OrderBy(x => x);
        var itmStrike = putStrikes.Last();
        var otmStrike = putStrikes.First();

        var optionStrategy = OptionStrategies.BearPutSpread(_symbol, itmStrike, otmStrike, expiry);
        Buy(optionStrategy, 1);
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm
IndexOptionBearPutSpreadAlgorithm.cs C#

Option Strategies

Bull Call Spread

Introduction

Bull call spread , also known as long call spread , consists of buying an ITM call and selling an OTM call. Both calls have the same underlying Equity and the same expiration date. The OTM call serves as a hedge for the ITM call. The bull call spread profits from a rise in underlying asset price.

Implementation

Follow these steps to implement the bull call spread strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(500000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().CallSpread(30, 5));
}
```

   The `CallSpread` filter narrows the universe down to just the two contracts you need to form a bull call spread.

2. In the `OnData` method, select the expiration and strikes of the contracts in the strategy legs.

```
C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select the call Option contracts with the furthest expiry
    var expiry = chain.Max(x => x.Expiry);
    var calls = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Call);
    if (calls.Count() == 0) return;

    // Select the ITM and OTM contract strike prices from the remaining contracts
    var strikes = calls.Select(x => x.Strike).ToList();
    var otmStrike = strikes.Max();
    var itmStrike = strikes.Min();
```

3. In the `OnData` method, select the contracts and place the orders.

   Approach A: Call the `OptionStrategies.BullCallSpread` method with the details of each leg and then pass the result to the `Buy` method.

```
C#
var optionStrategy = OptionStrategies.BullCallSpread(_symbol, itmStrike, otmStrike, expiry);
Buy(optionStrategy, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

```
C#
var itmCall = calls.Single(x => x.Strike == itmStrike);
var otmCall = calls.Single(x => x.Strike == otmStrike);

var legs = new List<Leg>()
    {
        Leg.Create(itmCall.Symbol, 1),
        Leg.Create(otmCall.Symbol, -1)
    };
ComboMarketOrder(legs, 1);
```

Strategy Payoff

The bull call spread is a limited-reward-limited-risk strategy. The payoff is

$$
\begin{aligned}
C_T^{OTM} &= (S_T - K^{OTM})^+ \\
C_T^{ITM} &= (S_T - K^{ITM})^+ \\
P_T &= (C_T^{ITM} - C_T^{OTM} + C_0^{OTM} - C_0^{ITM}) \times m - fee
\end{aligned}
$$

where C_T^{OTM} = OTM call value at time T

C_T^{ITM} = ITM call value at time T

S_T = Underlying asset price at time T

K^{OTM} = OTM call strike price

K^{ITM} = ITM call strike price

P_T = Payout total at time T

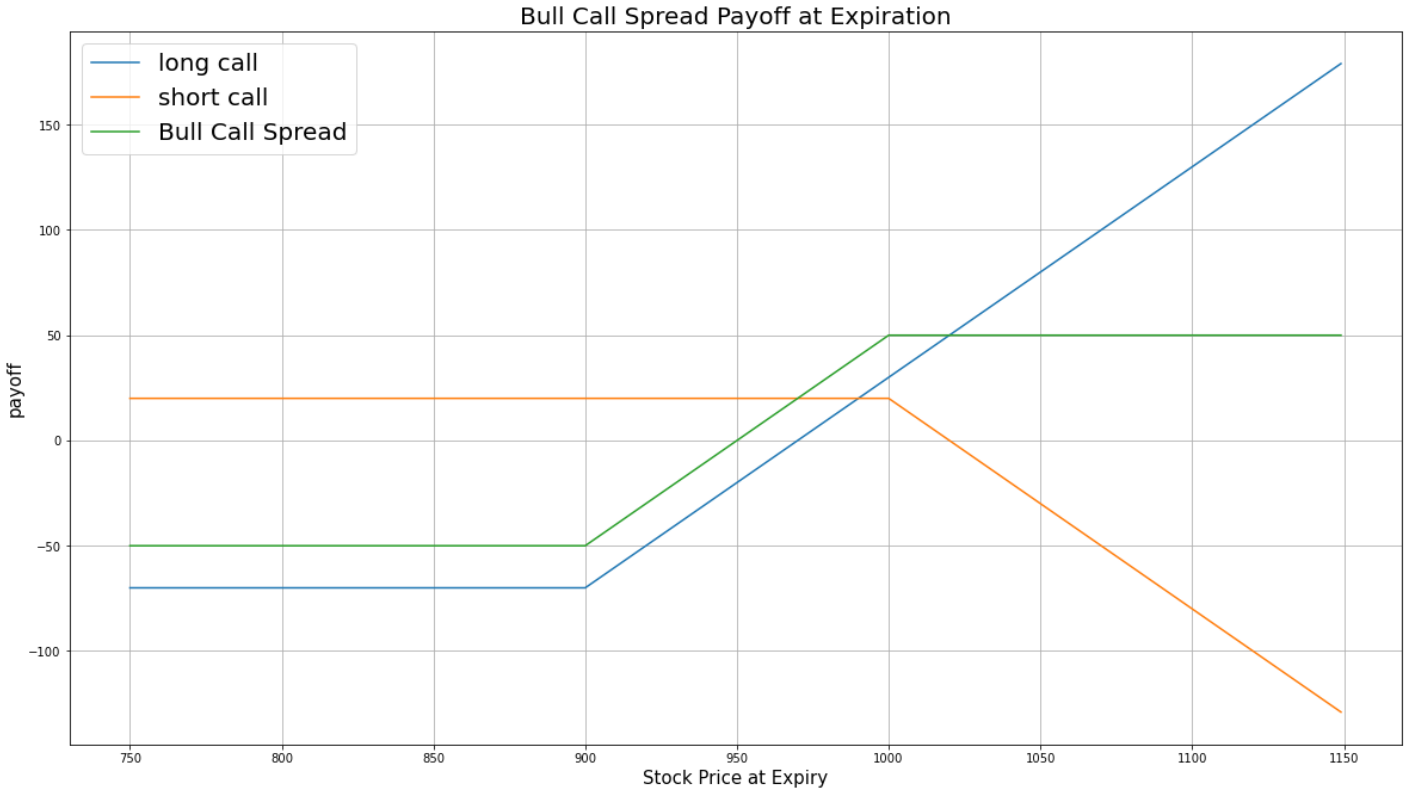
C_0^{ITM} = ITM call value at position opening (debit paid)

C_0^{OTM} = OTM call value at position opening (credit received)

m = Contract multiplier

T = Time of expiration

The following chart shows the payoff at expiration:



The maximum profit is $K^{OTM} - K^{ITM} + C_0^{OTM} - C_0^{ITM}$. If the underlying price increases to exceed both strikes at expiration, both calls are worth $(S_T - K)$ at expiration.

The maximum loss is the net debit you paid to open the position, $C_0^{OTM} - C_0^{ITM}$.

If the Option is American Option, there is a risk of early assignment on the contract you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	Price ($)	Strike ($)
OTM call	3.00	835.00
ITM call	41.00	767.50
Underlying Equity at expiration	829.08	-

Therefore, the payoff is

$$
\begin{aligned}
C_T^{OTM} &= (S_T - K^{OTM})^+ \\
&= (829.08 - 835.00)^+ \\
&= 0 \\
C_T^{ITM} &= (S_T - K^{ITM})^+ \\
&= (829.08 - 767.50)^+ \\
&= 61.58 \\
P_T &= (C_T^{ITM} - C_T^{OTM} + C_0^{OTM} - C_0^{ITM}) \times m - fee \\
&= (61.58 - 0 + 3.00 - 41.00) \times 100 - 1.00 \times 2 \\
&= 2356
\end{aligned}
$$

So, the strategy profits $2,356.

The following algorithm implements a bull call spread strategy:

```C#
public class BullCallSpreadExmapleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(500000);

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().CallSpread(30, 5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested) return;

        // Get the OptionChain
        var chain = slice.OptionChains.get(_symbol, null);
        if (chain.Count() == 0) return;

        // Get the furthest expiration date of the contracts
        var expiry = chain.OrderByDescending(x => x.Expiry).First().Expiry;

        // Select the call Option contracts with the furthest expiry
        var calls = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Call);
        if (calls.Count() == 0) return;

        // Select the ITM and OTM contract strikes from the remaining contracts
        var putStrikes = calls.Select(x => x.Strike).OrderBy(x => x);
        var itmStrike = putStrikes.First();
        var otmStrike = putStrikes.Last();

        var optionStrategy = OptionStrategies.BullCallSpread(_symbol, itmStrike, otmStrike, expiry);
        Buy(optionStrategy, 1);
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm
IndexOptionBullCallSpreadAlgorithm.cs C#

Option Strategies

Bull Put Spread

Introduction

Bull put spread , also known as long put spread , consists of buying an OTM put and selling an ITM put. Both puts have the same underlying Equity and the same expiration date. The OTM put serves as a hedge for the ITM put. The bull put spread profits from a rise in underlying asset price.

Implementation

Follow these steps to implement the bull put spread strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(500000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().PutSpread(30, 5));
}
```

   The `PutSpread` filter narrows the universe down to just the two contracts you need to form a bull put spread.

2. In the `OnData` method, select the expiration and strikes of the contracts in the strategy legs.

```
C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select the put Option contracts with the furthest expiry
    var expiry = chain.Max(x => x.Expiry);
    var puts = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Put);
    if (puts.Count() == 0) return;

    // Select the ITM and OTM contract strike prices from the remaining contracts
    var strikes = puts.Select(x => x.Strike).ToList();
    var otmStrike = strikes.Min();
    var itmStrike = strikes.Max();
```

3. In the `OnData` method, select the contracts and place the orders.

   Approach A: Call the `OptionStrategies.BullPutSpread` method with the details of each leg and then pass the result to the `Buy` method.

```C#
var optionStrategy = OptionStrategies.BullPutSpread(_symbol, itmStrike, otmStrike, expiry);
Buy(optionStrategy, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

```C#
var itmPut = puts.Single(x => x.Strike == itmStrike);
var otmPut = puts.Single(x => x.Strike == otmStrike);

var legs = new List<Leg>()
    {
        Leg.Create(itmPut.Symbol, -1),
        Leg.Create(otmPut.Symbol, 1)
    };
ComboMarketOrder(legs, 1);
```

Strategy Payoff

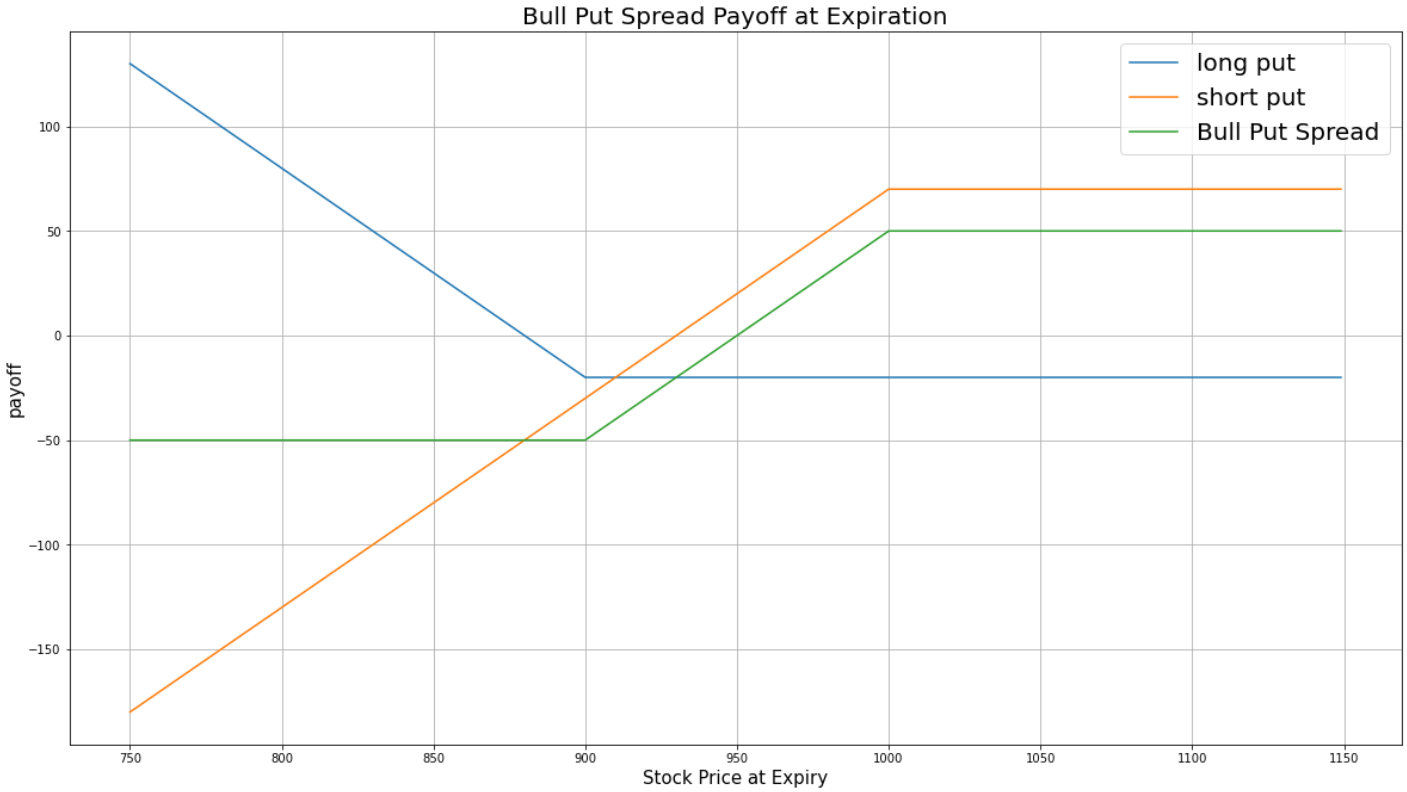
This is a limited-reward-limited-risk strategy. The payoff is

$$
\begin{aligned}
P_T^{OTM} &= (K^{OTM} - S_T)^+ \\
P_T^{ITM} &= (K^{ITM} - S_T)^+ \\
P_T &= (P_T^{OTM} - P_T^{ITM} + P_0^{ITM} - P_0^{OTM}) \times m - fee
\end{aligned}
$$

where

P_T^{OTM}	=	OTM put value at time T
P_T^{ITM}	=	ITM put value at time T
S_T	=	Underlying asset price at time T
K^{OTM}	=	OTM put strike price
K^{ITM}	=	ITM put strike price
P_T	=	Payout total at time T
P_0^{ITM}	=	ITM put value at position opening (credit received)
P_0^{OTM}	=	OTM put value at position opening (debit paid)
m	=	Contract multiplier
T	=	Time of expiration

The following chart shows the payoff at expiration:



The maximum profit is the net credit you received when opening the position, $P_0^{ITM} - P_0^{OTM}$. If the underlying price is higher than the strike prices of both put contracts at expiration, both puts expire worthless.

The maximum loss is $K^{OTM} - K^{ITM} + P_0^{ITM} - P_0^{OTM}$.

If the Option is American Option, there is a risk of early assignment on the contract you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	**Price ($)**	**Strike ($)**
OTM put	5.70	767.50
ITM put	35.50	835.00
Underlying Equity at expiration	829.08	-

Therefore, the payoff is

$$
\begin{aligned}
P_T^{OTM} &= (K^{OTM} - S_T)^+ \\
&= (767.50 - 829.08)^+ \\
&= 0 \\
P_T^{ITM} &= (K^{ITM} - S_T)^+ \\
&= (835.00 - 829.08)^+ \\
&= 5.92 \\
P_T &= (P_T^{OTM} - P_T^{ITM} + P_0^{ITM} - P_0^{OTM}) \times m - fee \\
&= (0 - 5.92 + 35.50 - 5.70) \times 100 - 1.00 \times 2 \\
&= 2386
\end{aligned}
$$

So, the strategy profits $2,386.

The following algorithm implements a bull put spread strategy:

```C#
public class BullPutSpreadExmapleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(500000);

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().PutSpread(30, 5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested) return;

        // Get the OptionChain
        var chain = slice.OptionChains.get(_symbol, null);
        if (chain.Count() == 0) return;

        // Get the furthest expiration date of the contracts
        var expiry = chain.OrderByDescending(x => x.Expiry).First().Expiry;

        // Select the put Option contracts with the furthest expiry
        var puts = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Put);
        if (puts.Count() == 0) return;

        // Select the ITM and OTM contract strikes from the remaining contracts
        var putStrikes = puts.Select(x => x.Strike).OrderBy(x => x);
        var itmStrike = putStrikes.Last();
        var otmStrike = putStrikes.First();

        var optionStrategy = OptionStrategies.BullPutSpread(_symbol, itmStrike, otmStrike, expiry);
        Buy(optionStrategy, 1);
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm
IndexOptionBullPutSpreadAlgorithm.cs C#

Option Strategies

Long Call Butterfly

Introduction

The long call butterfly strategy is the combination of a bull call spread and a bear call spread . In the call butterfly, all of the calls should have the same underlying Equity, the same expiration date, and the same strike price distance between the ITM-ATM and OTM-ATM call pairs. The long call butterfly consists of a long ITM call, a long OTM call, and 2 short ATM calls. This strategy profits from low volatility in the underlying Equity price.

Implementation

Follow these steps to implement the long call butterfly strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(500000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().CallButterfly(30, 5));
}
```

   The `CallSpread` filter narrows the universe down to just the three contracts you need to form a long call butterfly.

2. In the `OnData` method, select the contracts of the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select the call Option contracts with the furthest expiry
    var expiry = chain.Max(x =x> x.Expiry);
    var calls = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Call);
    if (calls.Count() == 0) return;

    // Select the ATM, ITM and OTM contracts from the remaining contracts
    var atmCall = calls.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First();
    var itmCall = calls.OrderBy(x => x.Strike).SkipLast(1).Last();
    var otmCall = calls.Single(x => x.Strike == atmCall.Strike * 2 - itmCall.Strike);
```

3. In the `OnData` method, place the orders.

   Approach A: Call the `OptionStrategies.ButterflyCall` method with the details of each leg and then pass the result to the `Buy` method.

```
var optionStrategy = OptionStrategies.ButterflyCall(_symbol, itmCall.Strike, atmCall.Strike, otmCall.Strike,
expiry);
Buy(optionStrategy, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

```
var legs = new List<Leg>()
    {
        Leg.Create(atmCall.Symbol, -2),
        Leg.Create(itmCall.Symbol, 1),
        Leg.Create(otmCall.Symbol, 1)
    };
ComboMarketOrder(legs, 1);
```

Strategy Payoff

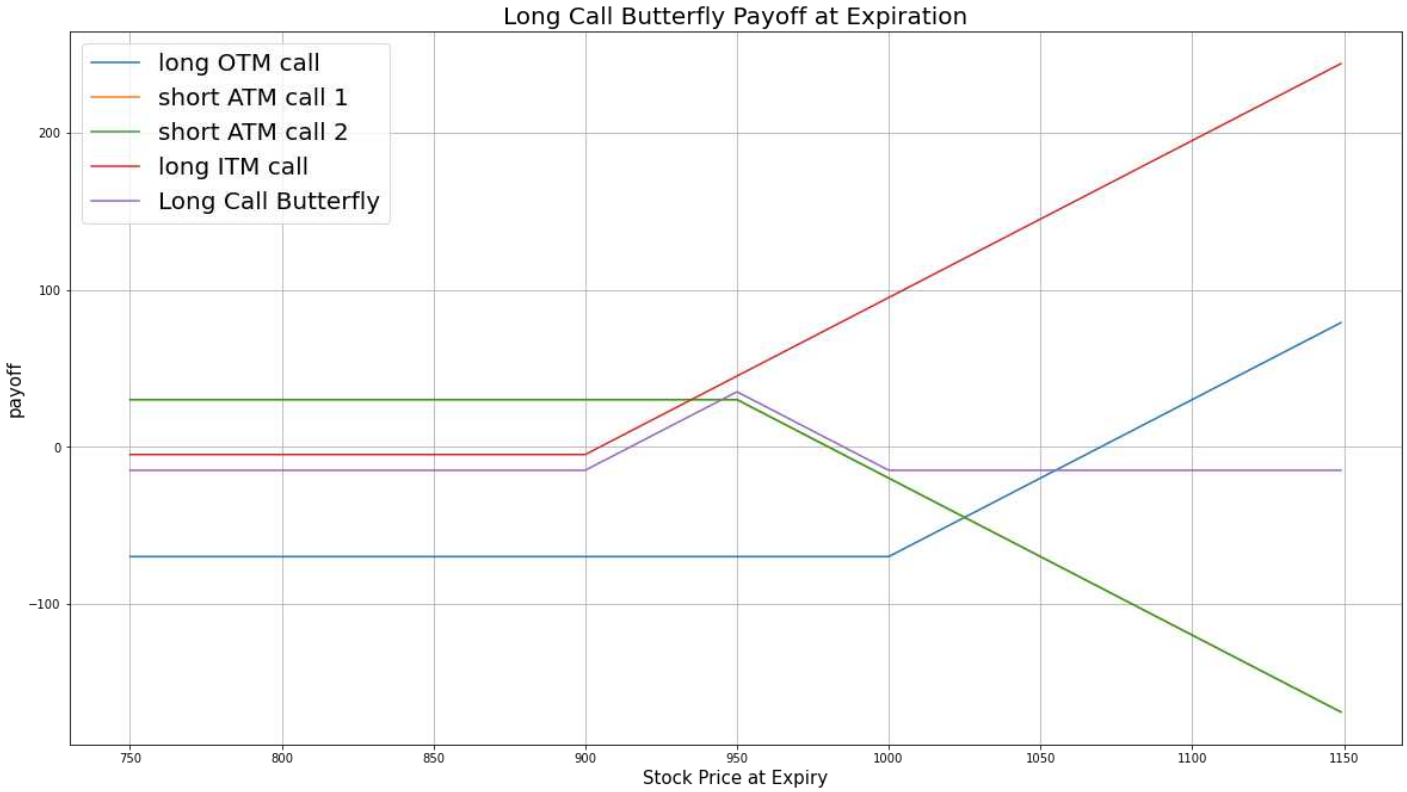
The long call butterfly is a limited-reward-limited-risk strategy. The payoff is

$$
\begin{aligned}
C_T^{OTM} &= (S_T - K^{OTM})^+ \\
C_T^{ITM} &= (S_T - K^{ITM})^+ \\
C_T^{ATM} &= (S_T - K^{ATM})^+ \\
P_T &= (C_T^{OTM} + C_T^{ITM} - 2 \times C_T^{ATM} + 2 \times C_0^{ATM} - C_0^{ITM} - C_0^{OTM}) \times m - fee
\end{aligned}
$$

where

C_T^{OTM}	=	OTM call value at time T
C_T^{ITM}	=	ITM call value at time T
C_T^{ATM}	=	ATM call value at time T
S_T	=	Underlying asset price at time T
K^{OTM}	=	OTM call strike price
K^{ITM}	=	ITM call strike price
K^{ATM}	=	ATM call strike price
P_T	=	Payout total at time T
C_0^{ITM}	=	ITM call value at position opening (debit paid)
C_0^{OTM}	=	OTM call value at position opening (debit paid)
C_0^{ATM}	=	OTM call value at position opening (credit received)
m	=	Contract multiplier
T	=	Time of expiration

The following chart shows the payoff at expiration:



The maximum profit is $K^{ATM} - K^{ITM} + 2 \times C_0^{ATM} - C_0^{ITM} - C_0^{OTM}$. It occurs when the underlying price is the same price at expiration as it was when opening the position and the payouts of the bull and bear call spreads are at their maximum.

The maximum loss is the net debit paid: $2 \times C_0^{ATM} - C_0^{ITM} - C_0^{OTM}$. It occurs when the underlying price is less than ITM strike or greater than OTM strike at expiration.

If the Option is American Option, there is a risk of early assignment on the contracts you sell.

Example

The following table shows the price details of the assets in the long call butterfly:

Asset	**Price ($)**	**Strike ($)**
OTM call	4.90	767.50
ATM call	15.00	800.00
ITM call	41.00	832.50
Underlying Equity at expiration	829.08	-

Therefore, the payoff is

$$
\begin{aligned}
C_T^{OTM} &= (S_T - K^{OTM})^+ \\
&= (767.50 - 829.08)^+ \\
&= 0 \\
C_T^{ITM} &= (S_T - K^{ITM})^+ \\
&= (832.50 - 829.08)^+ \\
&= 3.42 \\
C_T^{ATM} &= (S_T - K^{ATM})^+ \\
&= (800.00 - 829.08)^+ \\
&= 0 \\
P_T &= (C_T^{OTM} + C_T^{ITM} - 2 \times C_T^{ATM} + 2 \times C_0^{ATM} - C_0^{ITM} - C_0^{OTM}) \times m - fee \\
&= (0 + 3.42 - 0 \times 2 - 4.90 - 41.00 + 15.00 \times 2) \times 100 - 1.00 \times 4 \\
&= -1252
\end{aligned}
$$

So, the strategy loses $1,252.

The following algorithm implements a long call butterfly Option strategy:

```C#
public class BearPutSpreadStrategy : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(500000);

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().CallButterfly(30, 5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested) return;

        // Get the OptionChain of the symbol
        var chain = slice.OptionChains.get(_symbol, null);
        if (chain == null || chain.Count() == 0) return;

        // sorted the optionchain by expiration date and choose the furthest date
        var expiry = chain.OrderByDescending(x => x.Expiry).First().Expiry;

        // filter the call options from the contracts which expire on the furthest expiration date in the option
chain.
        var calls = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Call);
        if (calls.Count() == 0) return;

        // sort the call options with the same expiration date according to their strike price.
        var callStrikes = calls.Select(x => x.Strike).OrderBy(x => x);

        // get at-the-money strike
        var atmStrike = calls.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First().Strike;

        // Get the distance between lowest strike price and ATM strike, and highest strike price and ATM strike.
        // Get the lower value as the spread distance as equidistance is needed for both side.
        var spread = Math.Min(Math.Abs(callStrikes.First() - atmStrike), Math.Abs(callStrikes.Last() - atmStrike));

        // select the strike prices for forming the option legs
        var itmStrike = atmStrike - spread;
        var otmStrike = atmStrike + spread;

        var optionStrategy = OptionStrategies.CallButterfly(_symbol, otmStrike, atmStrike, itmStrike, expiry);
        // We open a position with 1 unit of the option strategy
        Buy(optionStrategy, 1);    // if long call butterfly
        //Sell(optionStrategy, 1);   // if short call butterfly
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm
IndexOptionCallButterflyAlgorithm.cs C#

Option Strategies

Short Call Butterfly

Introduction

The short call butterfly strategy is the combination of a bull call spread and a bear call spread . In the call butterfly, all of the calls should have the same underlying Equity, the same expiration date, and the same strike price distance between the ITM-ATM and OTM-ATM call pairs. The short call butterfly consists of a short ITM call, a short OTM call, and 2 long ATM calls. This strategy profits from high volatility in the underlying Equity price.

Implementation

Follow these steps to implement the short call butterfly strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(500000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().CallButterfly(30, 5));
}
```

   The `CallButterfly` filter narrows the universe down to just the three contracts you need to form a short call butterfly.

2. In the `OnData` method, select the contracts of the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select the call Option contracts with the furthest expiry
    var expiry = chain.Max(x =x> x.Expiry);
    var calls = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Call);
    if (calls.Count() == 0) return;

    // Select the ATM, ITM and OTM contracts from the remaining contracts
    var atmCall = calls.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First();
    var itmCall = calls.OrderBy(x => x.Strike).Skip(1).First();
    var otmCall = calls.Single(x => x.Strike == atmCall.Strike * 2 - itmCall.Strike);
```

3. In the `OnData` method, place the orders.

   Approach A: Call the `OptionStrategies.ShortButterflyCall` method with the details of each leg and then pass the result to the `Buy` method.

```
var optionStrategy = OptionStrategies.ShortButterflyCall(_symbol, otmCall.Strike, atmCall.Strike,
itmCall.Strike, expiry);
Buy(optionStrategy, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

```
var legs = new List<Leg>()
    {
        Leg.Create(atmCall.Symbol, 2),
        Leg.Create(itmCall.Symbol, -1),
        Leg.Create(otmCall.Symbol, -1)
    };
ComboMarketOrder(legs, 1);
```

If the Option is American Option, there is a risk of early assignment on the contracts you sell.

Strategy Payoff

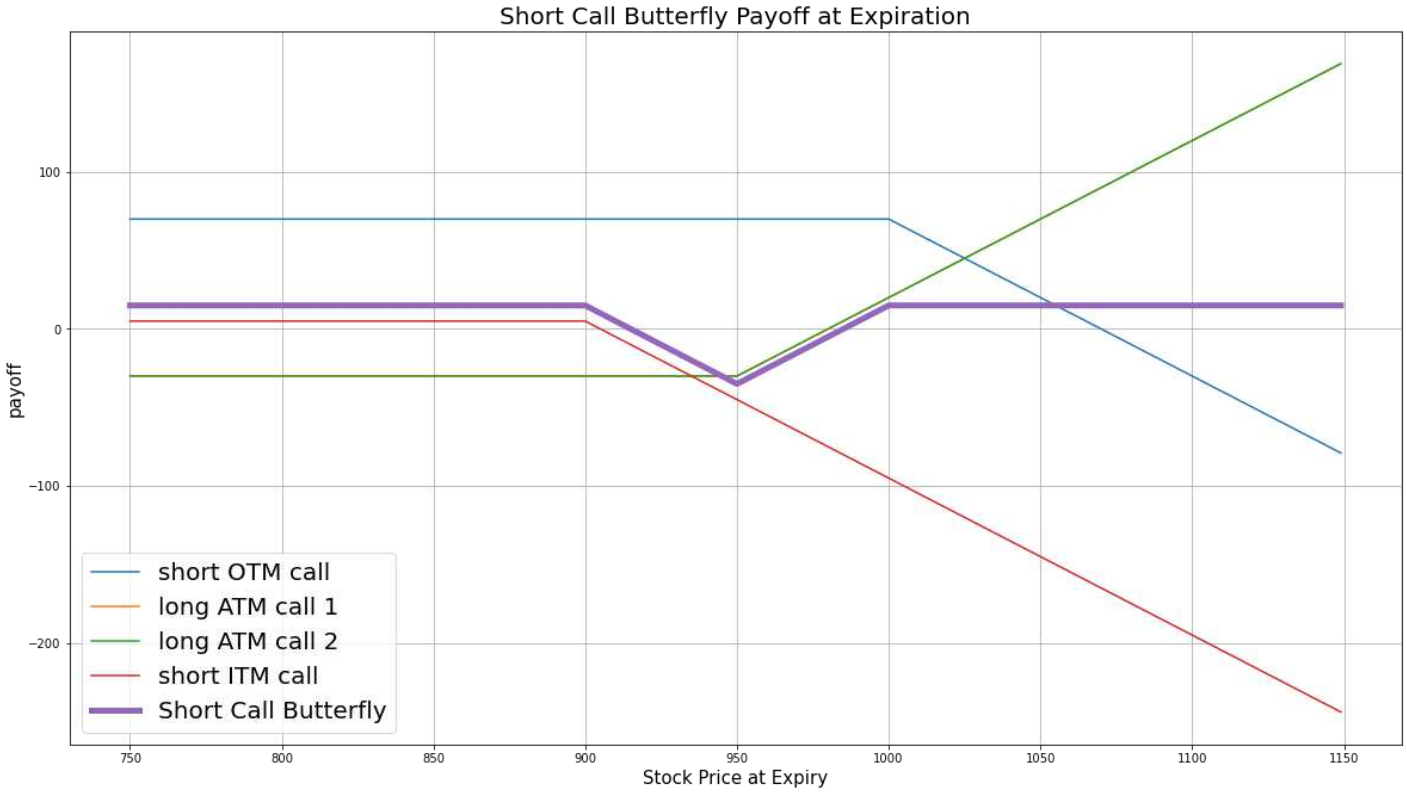
The short call butterfly is a limited-reward-limited-risk strategy. The payoff is

$$
\begin{aligned}
C_T^{OTM} &= (S_T - K^{OTM})^+ \\
C_T^{ITM} &= (S_T - K^{ITM})^+ \\
C_T^{ATM} &= (S_T - K^{ATM})^+ \\
P_T &= (2 \times C_T^{ATM} - C_T^{OTM} - C_T^{ITM} - 2 \times C_0^{ATM} + C_0^{ITM} + C_0^{OTM}) \times m - fee
\end{aligned}
$$

where

C_T^{OTM}	=	OTM call value at time T
C_T^{ITM}	=	ITM call value at time T
C_T^{ATM}	=	ATM call value at time T
S_T	=	Underlying asset price at time T
K^{OTM}	=	OTM call strike price
K^{ITM}	=	ITM call strike price
K^{ATM}	=	ATM call strike price
P_T	=	Payout total at time T
C_0^{ITM}	=	ITM call value at position opening (credit received)
C_0^{OTM}	=	OTM call value at position opening (credit received)
C_0^{ATM}	=	ATM call value at position opening (debit paid)
m	=	Contract multiplier
T	=	Time of expiration

The following chart shows the payoff at expiration:



The maximum profit is the net credit received: $C_0^{ITM} + C_0^{OTM} - 2 \times C_0^{ATM}$. It occurs when the underlying price is less than ITM strike or greater than OTM strike at expiration.

The maximum loss is $K^{ATM} - K^{ITM} + C_0^{ITM} + C_0^{OTM} - 2 \times C_0^{ATM}$. It occurs when the underlying price is at the same level as when you opened the trade.

If the Option is American Option, there is a risk of early assignment on the contracts you sell.

Example

The following table shows the price details of the assets in the short call butterfly:

Asset	**Price ($)**	**Strike ($)**
OTM call	4.90	767.50
ATM call	15.00	800.00
ITM call	41.00	832.50
Underlying Equity at expiration	829.08	-

Therefore, the payoff is

$$
\begin{aligned}
C_T^{OTM} &= (S_T - K^{OTM})^+ \\
&= (767.50 - 829.08)^+ \\
&= 0 \\
C_T^{ITM} &= (S_T - K^{ITM})^+ \\
&= (832.50 - 829.08)^+ \\
&= 3.42 \\
C_T^{ATM} &= (S_T - K^{ATM})^+ \\
&= (800.00 - 829.08)^+ \\
&= 0 \\
P_T &= (-C_T^{OTM} - C_T^{ITM} + 2 \times C_T^{ATM} - 2 \times C_0^{ATM} + C_0^{ITM} + C_0^{OTM}) \times m - fee \\
&= (-0 - 3.42 + 0 \times 2 + 4.90 + 41.00 - 15.00 \times 2) \times 100 - 1.00 \times 4 \\
&= -1252
\end{aligned}
$$

So, the strategy gains $1,244.

The following algorithm implements a short call butterfly Option strategy:

```C#
public class BearPutSpreadStrategy : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(500000);

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().CallButterfly(30, 5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested) return;

        // Get the OptionChain of the symbol
        var chain = slice.OptionChains.get(_symbol, null);
        if (chain == null || chain.Count() == 0) return;

        // sorted the optionchain by expiration date and choose the furthest date
        var expiry = chain.OrderByDescending(x => x.Expiry).First().Expiry;

        // filter the call options from the contracts which expire on the furthest expiration date in the option
chain.
        var calls = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Call);
        if (calls.Count() == 0) return;

        // sort the call options with the same expiration date according to their strike price.
        var callStrikes = calls.Select(x => x.Strike).OrderBy(x => x);

        // get at-the-money strike
        var atmStrike = calls.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First().Strike;

        // Get the distance between lowest strike price and ATM strike, and highest strike price and ATM strike.
        // Get the lower value as the spread distance as equidistance is needed for both side.
        var spread = Math.Min(Math.Abs(callStrikes.First() - atmStrike), Math.Abs(callStrikes.Last() - atmStrike));

        // select the strike prices for forming the option legs
        var itmStrike = atmStrike - spread;
        var otmStrike = atmStrike + spread;

        var optionStrategy = OptionStrategies.ShortButterflyCall(_symbol, otmStrike, atmStrike, itmStrike, expiry);
        // We open a position with 1 unit of the option strategy
        Buy(optionStrategy, 1);
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm
LongAndShortButterflyCallStrategiesAlgorithm.cs C#

Option Strategies

Long Put Butterfly

Introduction

Long Put butterfly is the combination of a bull put spread and a bear put spread . In this strategy, all the puts have the same underlying stock, the same expiration date, and the strike price distance of ITM-ATM and OTM-ATM put pairs are the same. The long put butterfly strategy consists of buying an ITM put, buying an OTM put, and selling 2 ATM puts. This strategy profits from low volatility.

Implementation

Follow these steps to implement the long put butterfly strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(500000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().PutButterfly(30, 5));
}
```

   The `PutButterfly` filter narrows the universe down to just the three contracts you need to form a long put butterfly.

2. In the `OnData` method, select the contracts of the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select the call Option contracts with the furthest expiry
    var expiry = chain.Max(x =x> x.Expiry);
    var puts = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Put);
    if (puts.Count() == 0) return;

    // Select the ATM, ITM and OTM contracts from the remaining contracts
    var atmPut = puts.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First();
    var itmPut = puts.OrderBy(x => x.Strike).SkipLast(1).Last();
    var otmPut = puts.Single(x => x.Strike == atmPut.Strike * 2 - itmPut.Strike);
```

3. In the `OnData` method, place the orders.

   Approach A: Call the `OptionStrategies.ButterflyPut` method with the details of each leg and then pass the result to the `Buy` method.

```
var optionStrategy = OptionStrategies.ButterflyPut(_symbol, itmPut.Strike, atmPut.Strike, otmPut.Strike,
expiry);
Buy(optionStrategy, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

```
var legs = new List<Leg>()
    {
        Leg.Create(atmPut.Symbol, -2),
        Leg.Create(itmPut.Symbol, 1),
        Leg.Create(otmPut.Symbol, 1)
    };
ComboMarketOrder(legs, 1);
```

Strategy Payoff

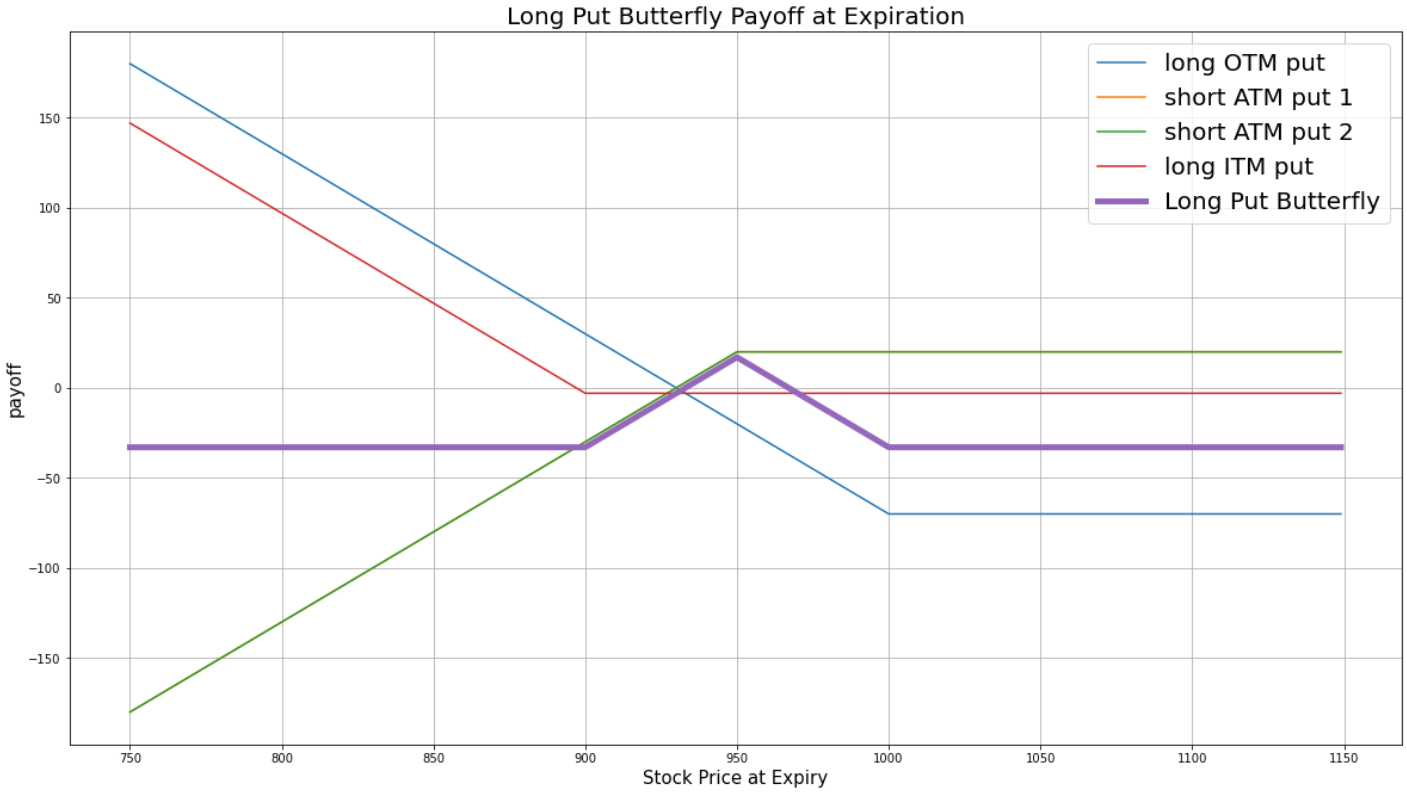
The long put butterfly is a limited-reward-limited-risk strategy. The payoff is

$$
\begin{aligned}
P_T^{OTM} &= (K^{OTM} - S_T)^+ \\
P_T^{ITM} &= (K^{ITM} - S_T)^+ \\
P_T^{ATM} &= (K^{ATM} - S_T)^+ \\
P_T &= (P_T^{OTM} + P_T^{ITM} - 2 \times P_T^{ATM} + 2 \times P_0^{ATM} - P_0^{ITM} - P_0^{OTM}) \times m - fee
\end{aligned}
$$

where

P_T^{OTM}	=	OTM put value at time T
P_T^{ITM}	=	ITM put value at time T
P_T^{ATM}	=	ATM put value at time T
S_T	=	Underlying asset price at time T
K^{OTM}	=	OTM put strike price
K^{ITM}	=	ITM put strike price
K^{ATM}	=	ATM put strike price
P_T	=	Payout total at time T
P_0^{ITM}	=	ITM put value at position opening (debit paid)
P_0^{OTM}	=	OTM put value at position opening (debit paid)
P_0^{ATM}	=	ATM put value at position opening (credit received)
m	=	Contract multiplier
T	=	Time of expiration

The following chart shows the payoff at expiration:



The maximum profit is $K^{ATM} - K^{OTM} + 2 \times P_0^{ATM} - P_0^{ITM} - P_0^{OTM}$. It occurs when the underlying price is the same at expiration as it was when you open the trade. In this case, the payout of the combined bull put and bear put spreads are at their maximum.

The maximum loss is the net debit paid, $2 \times P_0^{ATM} - P_0^{ITM} - P_0^{OTM}$. It occurs when the underlying price is below the ITM strike price or above the OTM strike price at expiration.

If the Option is American Option, there is a risk of early assignment on the contracts you sell.

Example

The following table shows the price details of the assets in the long put butterfly algorithm:

Asset	**Price ($)**	**Strike ($)**
ITM put	37.80	832.50
ATM put	14.70	800.00
OTM put	5.70	767.50
Underlying Equity at expiration	829.08	-

Therefore, the payoff is

$$
\begin{aligned}
P_T^{OTM} &= (K^{OTM} - S_T)^+ \\
&= (829.08 - 832.50)^+ \\
&= 0 \\
P_T^{ITM} &= (K^{ITM} - S_T)^+ \\
&= (829.08 - 767.50)^+ \\
&= 61.58 \\
P_T^{ATM} &= (K^{ATM} - S_T)^+ \\
&= (829.08 - 800.00)^+ \\
&= 29.08 \\
P_T &= (P_T^{OTM} + P_T^{ITM} - 2 \times P_T^{ATM} + 2 \times P_0^{ATM} - P_0^{ITM} - P_0^{OTM}) \times m - fee \\
&= (61.58 + 0 - 29.08 \times 2 - 5.70 - 37.80 + 14.70 \times 2) \times 100 - 1.00 \times 4 \\
&= -1072
\end{aligned}
$$

So, the strategy loses $1,072.

The following algorithm implements a long put butterfly Option strategy:

C#

```
public class BearPutSpreadStrategy : QCAlgorithm
{
    private Symbol _equity;
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        _equity = AddEquity("GOOG", Resolution.Minute).Symbol;
        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().PutButterfly(30, 5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested) return;

        // Get the OptionChain of the symbol
        var chain = slice.OptionChains.get(_symbol, null);
        if (chain == null || chain.Count() == 0) return;

        // sorted the optionchain by expiration date and choose the furthest date
        var expiry = chain.OrderByDescending(x => x.Expiry).First().Expiry;

        // filter the put options from the contracts which expire on the furthest expiration date in the option
chain.
        var puts = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Put);
        if (puts.Count() == 0) return;

        // sort the put options with the same expiration date according to their strike price.
        var putStrikes = puts.Select(x => x.Strike).OrderBy(x => x);

        // get at-the-money strike
        var atmStrike = puts.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First().Strike;

        // Get the distance between lowest strike price and ATM strike, and highest strike price and ATM strike.
        // Get the lower value as the spread distance as equidistance is needed for both side.
        var spread = Math.Min(Math.Abs(putStrikes.First() - atmStrike), Math.Abs(putStrikes.Last() - atmStrike));

        // select the strike prices for forming the option legs
        var itmStrike = atmStrike + spread;
        var otmStrike = atmStrike - spread;

        var optionStrategy = OptionStrategies.PutButterfly(_symbol, itmStrike, atmStrike, otmStrike, expiry);
        // We open a position with 1 unit of the option strategy
        Buy(optionStrategy, 1);    // if long put butterfly
        //Sell(optionStrategy, 1);   // if short put butterfly
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm
IndexOptionPutButterflyAlgorithm.cs C#

Option Strategies

Short Put Butterfly

Introduction

Short Put butterfly is the combination of a bull put spread and a bear put spread . In this strategy, all the puts have the same underlying stock, the same expiration date, and the strike price distance of ITM-ATM and OTM-ATM put pairs are the same. The short put butterfly strategy consists of selling an ITM put, selling an OTM put, and buying 2 ATM puts. This strategy profits from a drastic change in underlying price.

Implementation

Follow these steps to implement the short put butterfly strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(500000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().PutButterfly(30, 5));
}
```

   The `PutButterfly` filter narrows the universe down to just the three contracts you need to form a short put butterfly.

2. In the `OnData` method, select the contracts of the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select the call Option contracts with the furthest expiry
    var expiry = chain.Max(x =x> x.Expiry);
    var puts = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Put);
    if (puts.Count() == 0) return;

    // Select the ATM, ITM and OTM contracts from the remaining contracts
    var atmPut = puts.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First();
    var itmPut = puts.OrderBy(x => x.Strike).SkipLast(1).Last();
    var otmPut = puts.Single(x => x.Strike == atmPut.Strike * 2 - itmPut.Strike);
```

3. In the `OnData` method, place the orders.

   Approach A: Call the `OptionStrategies.ShortButterflyPut` method with the details of each leg and then pass the result to the `Buy` method.

```
var optionStrategy = OptionStrategies.ShortButterflyPut(_symbol, itmPut.Strike, atmPut.Strike, otmPut.Strike,
expiry);
Buy(optionStrategy, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

```
var legs = new List<Leg>()
    {
        Leg.Create(atmPut.Symbol, 2),
        Leg.Create(itmPut.Symbol, -1),
        Leg.Create(otmPut.Symbol, -1)
    };
ComboMarketOrder(legs, 1);
```

Strategy Payoff

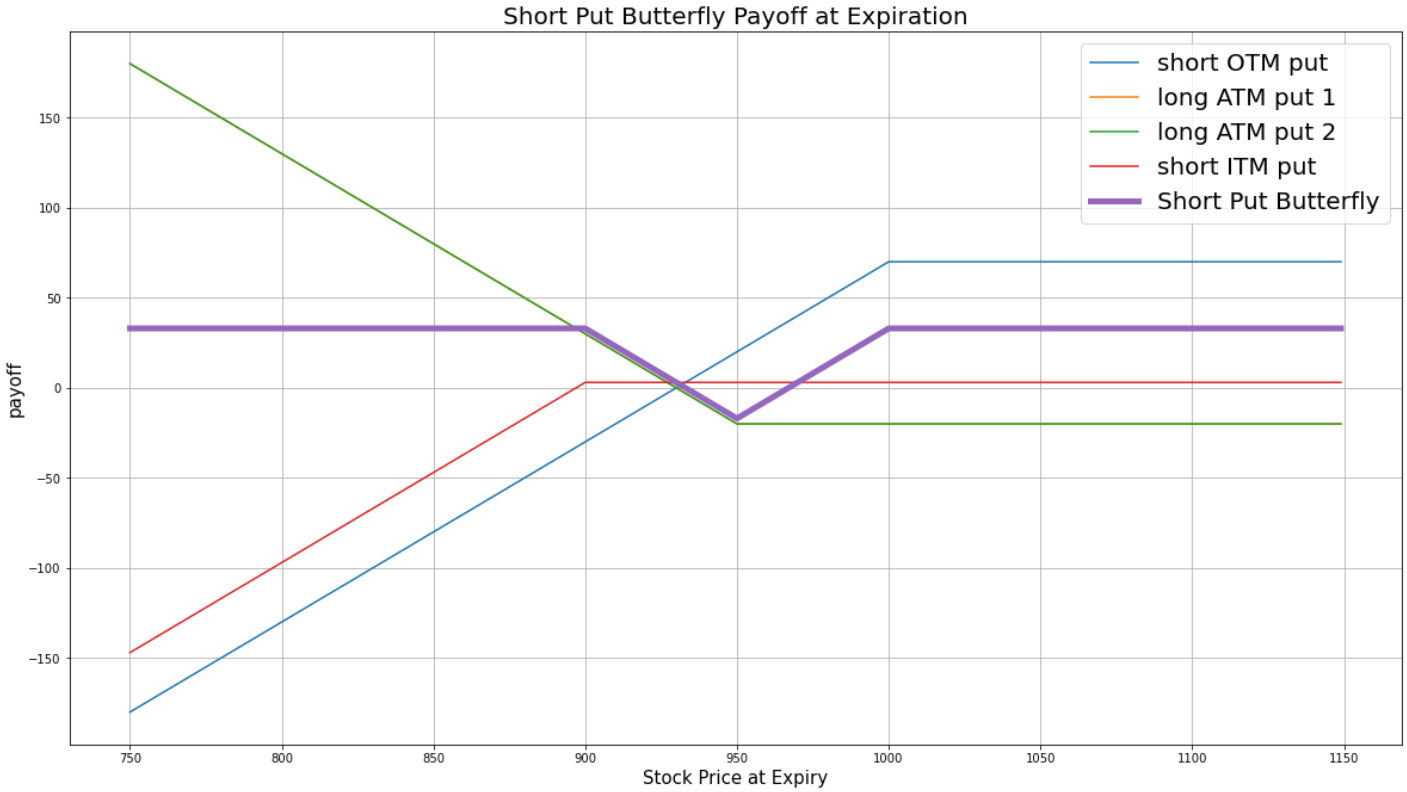
The short put butterfly is a limited-reward-limited-risk strategy. The payoff is

$$
\begin{aligned}
P_T^{OTM} &= (K^{OTM} - S_T)^+ \\
P_T^{ITM} &= (K^{ITM} - S_T)^+ \\
P_T^{ATM} &= (K^{ATM} - S_T)^+ \\
P_T &= (2 \times P_T^{ATM} - P_T^{OTM} - P_T^{ITM} - 2 \times P_0^{ATM} + P_0^{ITM} + P_0^{OTM}) \times m - fee
\end{aligned}
$$

where

P_T^{OTM}	=	OTM put value at time T
P_T^{ITM}	=	ITM put value at time T
P_T^{ATM}	=	ATM put value at time T
S_T	=	Underlying asset price at time T
K^{OTM}	=	OTM put strike price
K^{ITM}	=	ITM put strike price
K^{ATM}	=	ATM put strike price
P_T	=	Payout total at time T
P_0^{ITM}	=	ITM put value at position opening (credit received)
P_0^{OTM}	=	OTM put value at position opening (credit received)
P_0^{ATM}	=	ATM put value at position opening (debit paid)
m	=	Contract multiplier
T	=	Time of expiration

The following chart shows the payoff at expiration:



The maximum profit is the net credit received, $P_0^{ITM} + P_0^{OTM} - 2 \times P_0^{ATM}$. It occurs when the underlying price is below the ITM strike or above the OTM strike at expiration.

The maximum loss is $K^{ATM} - K^{OTM} - 2 \times P_0^{ATM} + P_0^{ITM} + P_0^{OTM}$. It occurs when the underlying price at expiration is at the same price as when you opened the trade.

If the Option is American Option, there is a risk of early assignment on the contracts you sell.

Example

The following table shows the price details of the assets in the short put butterfly algorithm:

Asset	**Price ($)**	**Strike ($)**
ITM put	37.80	832.50
ATM put	14.70	800.00
OTM put	5.70	767.50
Underlying Equity at expiration	829.08	-

Therefore, the payoff is

$$
\begin{aligned}
P_T^{OTM} &= (K^{OTM} - S_T)^+ \\
&= (829.08 - 832.50)^+ \\
&= 0 \\
P_T^{ITM} &= (K^{ITM} - S_T)^+ \\
&= (829.08 - 767.50)^+ \\
&= 61.58 \\
P_T^{ATM} &= (K^{ATM} - S_T)^+ \\
&= (829.08 - 800.00)^+ \\
&= 29.08 \\
P_T &= (-P_T^{OTM} - P_T^{ITM} + 2 \times P_T^{ATM} - 2 \times P_0^{ATM} + P_0^{ITM} + P_0^{OTM}) \times m - fee \\
&= (-61.58 - 0 + 29.08 \times 2 + 5.70 + 37.80 - 14.70 \times 2) \times 100 - 1.00 \times 4 \\
&= 1064
\end{aligned}
$$

So, the strategy gains $1,064.

The following algorithm implements a short put butterfly Option strategy:

```C#
public class BearPutSpreadStrategy : QCAlgorithm
{
    private Symbol _equity;
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        _equity = AddEquity("GOOG", Resolution.Minute).Symbol;
        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().PutButterfly(30, 5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested) return;

        // Get the OptionChain of the symbol
        var chain = slice.OptionChains.get(_symbol, null);
        if (chain == null || chain.Count() == 0) return;

        // sorted the optionchain by expiration date and choose the furthest date
        var expiry = chain.OrderByDescending(x => x.Expiry).First().Expiry;

        // filter the put options from the contracts which expire on the furthest expiration date in the option
chain.
        var puts = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Put);
        if (puts.Count() == 0) return;

        // sort the put options with the same expiration date according to their strike price.
        var putStrikes = puts.Select(x => x.Strike).OrderBy(x => x);

        // get at-the-money strike
        var atmStrike = puts.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First().Strike;

        // Get the distance between lowest strike price and ATM strike, and highest strike price and ATM strike.
        // Get the lower value as the spread distance as equidistance is needed for both side.
        var spread = Math.Min(Math.Abs(putStrikes.First() - atmStrike), Math.Abs(putStrikes.Last() - atmStrike));

        // select the strike prices for forming the option legs
        var itmStrike = atmStrike + spread;
        var otmStrike = atmStrike - spread;

        var optionStrategy = OptionStrategies.ShortButterflyPut(_symbol, itmStrike, atmStrike, otmStrike, expiry);
        // We open a position with 1 unit of the option strategy
        Buy(optionStrategy, 1);
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm
LongAndShortButterflyPutStrategiesAlgorithm.cs C#

Option Strategies

Long Call Calendar Spread

Introduction

Long call calendar spread , also known as call horizontal spread , is a combination of a longer-term (far-leg/front-month) call and a shorter-term (near-leg/back-month) call, where all calls have the same underlying stock and the same strike price. The long call calendar spread consists of buying a longer-term call and selling a shorter-term call. This strategy profits from a decrease in the underlying price. It also profits from the time decay value because the theta θ (the Option price decay by 1 day closer to maturity) of the shorter-term call is larger than longer-term call.

Implementation

Follow these steps to implement the long call calendar spread strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(500000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().CallCalendarSpread(0, 30, 60));
}
```

   The `CallCalendarSpread` filter narrows the universe down to just the two contracts you need to form a long call calendar spread.

2. In the `OnData` method, select the strike price and expiration dates of the contracts in the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Get the ATM strike
    var atmStrike = chain.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First().Strike;

    // Select the ATM call Option contracts
    var calls = chain.Where(x => x.Strike == atmStrike && x.Right == OptionRight.Call);
    if (calls.Count() == 0) return;

    // Select the near and far expiry contracts
    var expiries = calls.Select(x => x.Expiry).ToList();
    var nearExpiry = expiries.Min();
    var farExpiry = expiries.Max();
```

3. In the `OnData` method, select the contracts and place the orders.

Approach A: Call the `OptionStrategies.CallCalendarSpread` method with the details of each leg and then pass the result to the `Buy` method.

```C#
var optionStrategy = OptionStrategies.CallCalendarSpread(_symbol, atmStrike, nearExpiry, farExpiry);
Buy(optionStrategy, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

```C#
var nearExpiryCall = calls.Single(x => x.Expiry == nearExpiry);
var farExpiryCall = calls.Single(x => x.Expiry == farExpiry);

var legs = new List<Leg>()
    {
        Leg.Create(nearExpiryCall.Symbol, -1),
        Leg.Create(farExpiryCall.Symbol, 1)
    };
ComboMarketOrder(legs, 1);
```

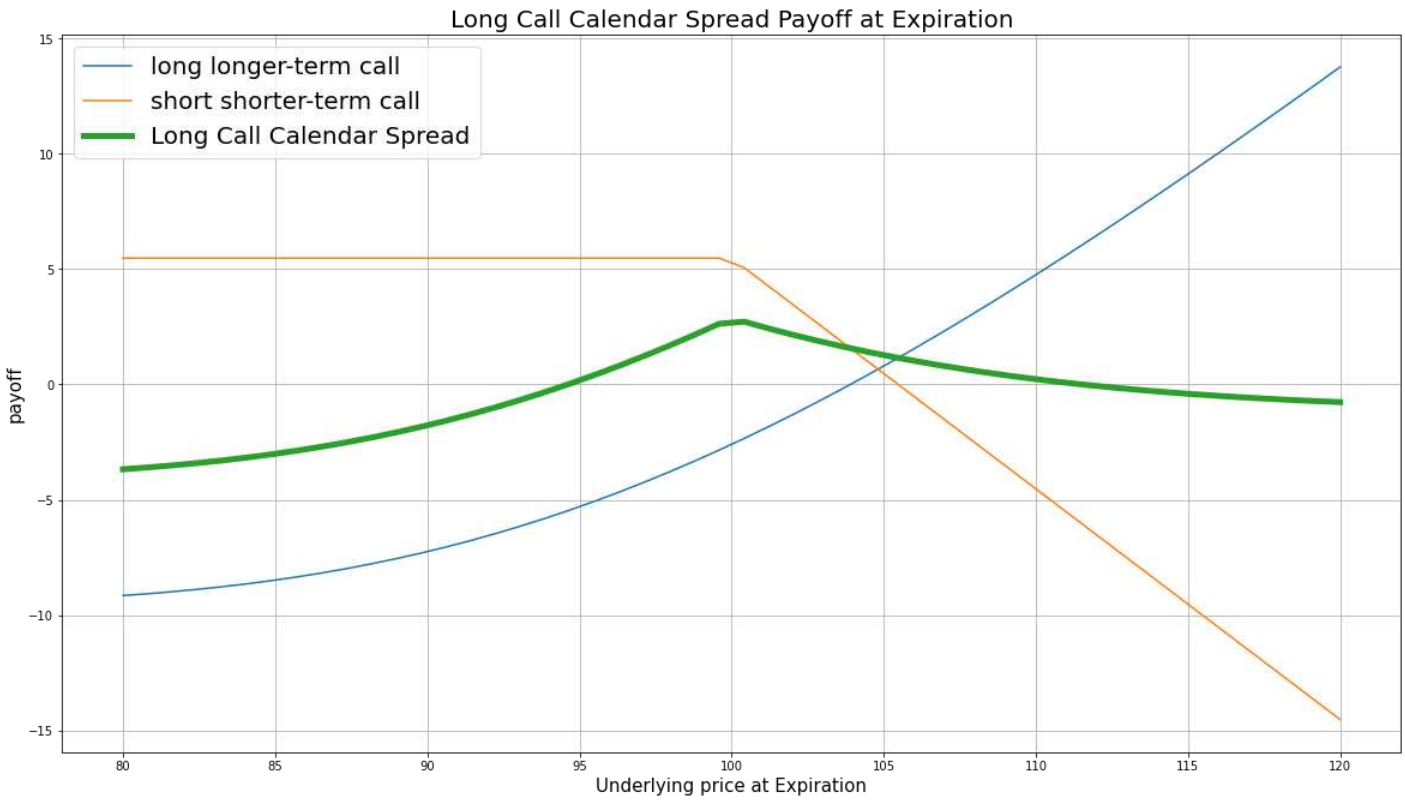
Strategy Payoff

The long call calendar spread is a limited-reward-limited-risk strategy. The payoff at the shorter-term expiration is

$$
\begin{aligned}
C_T^{\text{short-term}} &= (S_T - K)^+ \\
P_T &= (C_T^{\text{long-term}} - C_T^{\text{short-term}} + C_0^{\text{short-term}} - C_0^{\text{long-term}}) \times m - fee
\end{aligned}
$$

\begin{array}{rcll} \textrm{where} & C^{\textrm{short-term}}_T & = & \textrm{Shorter term call value at time T}\\ & C^{\textrm{long-term}}_T & = & \textrm{Longer term call value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K & = & \textrm{Strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & C^{\textrm{short-term}}_0 & = & \textrm{Shorter term call value at position opening (credit received)}\\ & C^{\textrm{long-term}}_0 & = & \textrm{Longer term call value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of shorter term call expiration} \end{array}

The following chart shows the payoff at expiration:



The maximum profit is undetermined because it depends on the underlying volatility. It occurs when $S_T = S_0$ and the spread of the calls are at their maximum.

The maximum loss is the net debit paid, $C^{\textrm{short-term}}_0 - C^{\textrm{long-term}}_0$. It occurs when the underlying price moves very deep ITM or OTM so the values of both calls are close to zero.

If the Option is American Option, there is a risk of early assignment on the contract you sell. If the buyer exercises the call you sell, you could lose all the debit you received if you don't close the long call and the underlying price drops below the long call strike price.

Example

The following table shows the price details of the assets in the long call calendar spread:

Asset	**Price ($)**	**Strike ($)**
Longer-term call at the start of the trade	4.40	835.00
Shorter-term call at the start of the trade	36.80	767.50
Longer-term call at time T	31.35	835.00
Underlying Equity at time T	829.08	-

Therefore, the payoff at time T (the expiration of the short-term call) is

$$\begin{array}{rcll} C^{\textrm{short-term}}_T & = & (S_T - K)^{+}\\ & = & (828.07-800.00)^{+}\\ & = & 28.07\\ P_T & = & (C^{\textrm{long-term}}_T - C^{\textrm{short-term}}_T + C^{\textrm{short-term}}_0 - C^{\textrm{long-term}}_0)\times m - fee\\ & = & (31.35-28.07+11.30-20.00)\times100-1.00\times2\\ & = & -544 \end{array}$$

So, the strategy loses $544.

The following algorithm implements a long call calendar spread Option strategy:

```C#
public class BearPutSpreadStrategy : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(500000);

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().CallCalendarSpread(0, 30, 60));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested) return;

        // Get the OptionChain of the symbol
        var chain = slice.OptionChains.get(_symbol, null);
        if (chain == null || chain.Count() == 0) return;

        // get at-the-money strike
        var atmStrike = chain.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First().Strike;

        // filter the call options from the contracts which is ATM in the option chain.
        var calls = chain.Where(x => x.Strike == atmStrike && x.Right == OptionRight.Call);
        if (calls.Count() == 0) return;

        // sorted the optionchain by expiration date
        var expiries = calls.Select(x => x.Expiry).OrderBy(x => x);

        // select the farest expiry as far-leg expiry, and the nearest expiry as near-leg expiry
        var nearExpiry = expiries.First();
        var farExpiry = expiries.Last();

        var optionStrategy = OptionStrategies.CallCalendarSpread(_symbol, atmStrike, nearExpiry, farExpiry);
        // We open a position with 1 unit of the option strategy
        Buy(optionStrategy, 1);        // if long call calendar spread
        //Sell(optionStrategy, 1);     // if short call calendar spread
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm
IndexOptionCallCalendarSpreadAlgorithm.cs C# LongAndShortCallCalendarSpreadStrategiesAlgorithm.cs C#

Option Strategies

Short Call Calendar Spread

Introduction

Call calendar spread , also known as call horizontal spread , is a combination of a longer-term (far-leg/front-month) call and a shorter-term (near-leg/back-month) call, where all calls have the same underlying stock and the same strike price. The short call calendar spread consists of selling a longer-term call and buying a shorter-term call. The strategy profits from from an increase in the underlying price.

Implementation

Follow these steps to implement the short call calendar spread strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(500000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().CallCalendarSpread(0, 30, 60));
}
```

   The `CallCalendarSpread` filter narrows the universe down to just the two contracts you need to form a short call calendar spread.

2. In the `OnData` method, select the strike price and expiration dates of the contracts in the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Get the ATM strike
    var atmStrike = chain.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First().Strike;

    // Select the ATM call Option contracts
    var calls = chain.Where(x => x.Strike == atmStrike && x.Right == OptionRight.Call);
    if (calls.Count() == 0) return;

    // Select the near and far expiry contracts
    var expiries = calls.Select(x => x.Expiry).ToList();
    var nearExpiry = expiries.Min();
    var farExpiry = expiries.Max();
```

3. In the `OnData` method, select the contracts and place the orders.

Approach A: Call the `OptionStrategies.ShortCallCalendarSpread` method with the details of each leg and then pass the result to the `Buy` method.

```
C#
var optionStrategy = OptionStrategies.ShortCallCalendarSpread(_symbol, atmStrike, nearExpiry, farExpiry);
Buy(optionStrategy, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

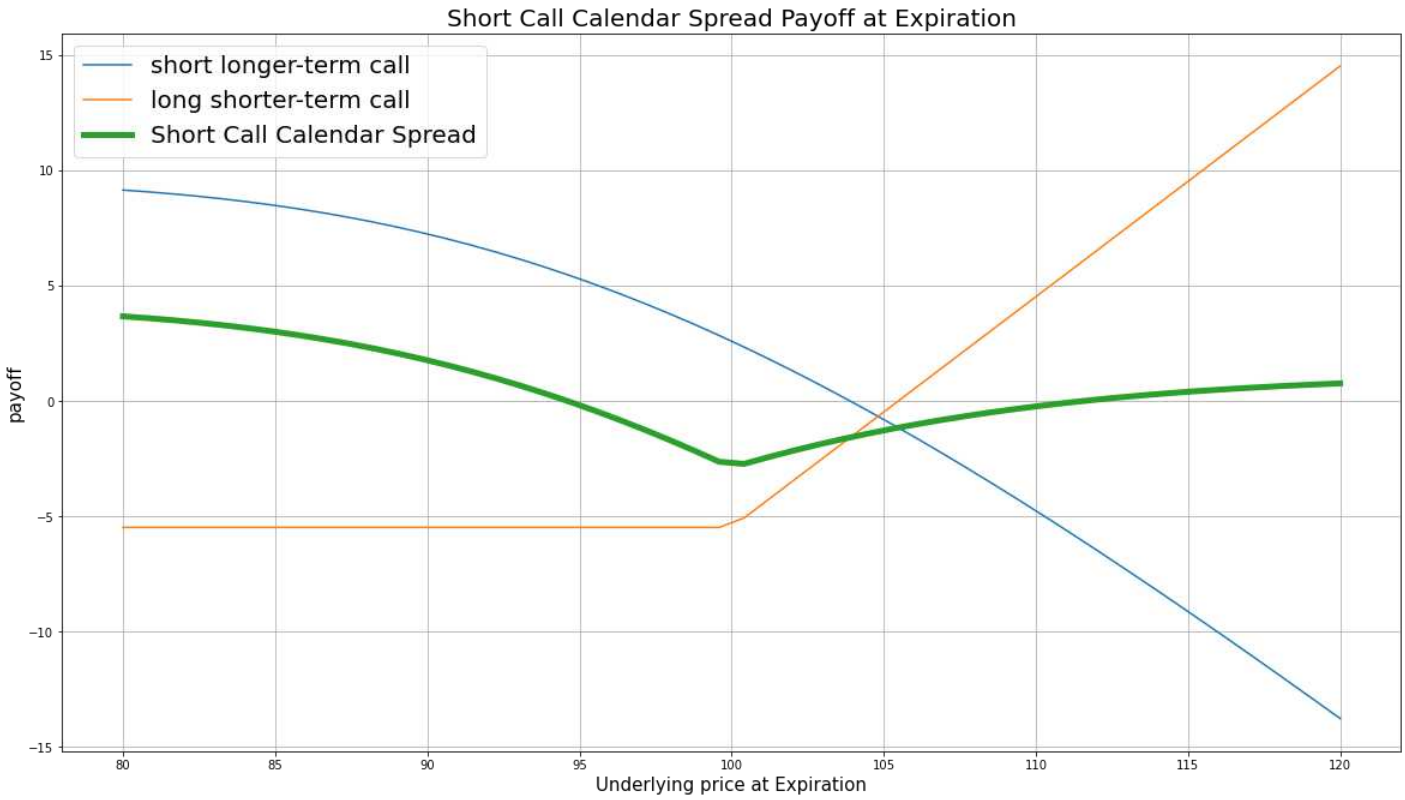
```
C#
var nearExpiryCall = calls.Single(x => x.Expiry == nearExpiry);
var farExpiryCall = calls.Single(x => x.Expiry == farExpiry);

var legs = new List<Leg>()
    {
        Leg.Create(nearExpiryCall.Symbol, 1),
        Leg.Create(farExpiryCall.Symbol, -1)
    };
ComboMarketOrder(legs, 1);
```

Strategy Payoff

The short call calendar spread is a limited-reward-limited-risk strategy. The payoff at the shorter-term expiration is

$$\begin{array}{rcll} C^{\textrm{short-term}}_T & = & (S_T - K)^{+}\\ P_T & = & (C^{\textrm{short-term}}_T - C^{\textrm{long-term}}_T + C^{\textrm{long-term}}_0 - C^{\textrm{short-term}}_0)\times m - fee \end{array} \begin{array}{rcl} \textrm{where} & & \\ C^{\textrm{short-term}}_T & = & \textrm{Shorter term call value at time T}\\ C^{\textrm{long-term}}_T & = & \textrm{Longer term call value at time T}\\ S_T & = & \textrm{Underlying asset price at time T}\\ K & = & \textrm{Strike price}\\ P_T & = & \textrm{Payout total at time T}\\ C^{\textrm{short-term}}_0 & = & \textrm{Shorter term call value at position opening (debit paid)}\\ C^{\textrm{long-term}}_0 & = & \textrm{Longer term call value at position opening (credit received)}\\ m & = & \textrm{Contract multiplier}\\ T & = & \textrm{Time of shorter term call expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is the net credit received, $C^{\textrm{long-term}}_0 - C^{\textrm{short-term}}_0$. It occurs when the underlying price moves very deep ITM or OTM so the values of both calls are close to zero.

The maximum loss is undetermined because it depends on the underlying volatility. It occurs when $S_T = S_0$ and the spread of the 2 calls are at their maximum.

If the Option is American Option, there is a risk of early assignment on the contract you sell. If you don't close the call positions together, the naked short call will have unlimited drawdown risk after the long call expires.

Example

The following table shows the price details of the assets in the short call calendar spread:

Asset	**Price ($)**	**Strike ($)**
Longer-term call at the start of the trade	4.40	835.00
Shorter-term call at the start of the trade	36.80	767.50
Longer-term call at time T	31.35	835.00
Underlying Equity at time T	829.08	-

Therefore, the payoff at time T (the expiration of the short-term call) is

$$\begin{array}{rcll} C^{\textrm{short-term}}_T & = & (S_T - K)^{+}\\ & = & (828.07-800.00)^{+}\\ & = & 28.07\\ P_T & = & (-C^{\textrm{long-term}}_T + C^{\textrm{short-term}}_T - C^{\textrm{short-term}}_0 + C^{\textrm{long-term}}_0)\times m - fee\\ & = & (-31.35+28.07-11.30+20.00)\times100-1.00\times2\\ & = & 540 \end{array}$$

So, the strategy gains $540.

The following algorithm implements a short call calendar spread Option strategy:

```C#
public class BearPutSpreadStrategy : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(500000);

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().CallCalendarSpread(0, 30, 60));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested) return;

        // Get the OptionChain of the symbol
        var chain = slice.OptionChains.get(_symbol, null);
        if (chain == null || chain.Count() == 0) return;

        // get at-the-money strike
        var atmStrike = chain.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First().Strike;

        // filter the call options from the contracts which is ATM in the option chain.
        var calls = chain.Where(x => x.Strike == atmStrike && x.Right == OptionRight.Call);
        if (calls.Count() == 0) return;

        // sorted the optionchain by expiration date
        var expiries = calls.Select(x => x.Expiry).OrderBy(x => x);

        // select the farest expiry as far-leg expiry, and the nearest expiry as near-leg expiry
        var nearExpiry = expiries.First();
        var farExpiry = expiries.Last();

        var optionStrategy = OptionStrategies.ShortCallCalendarSpread(_symbol, atmStrike, nearExpiry, farExpiry);
        // We open a position with 1 unit of the option strategy
        Buy(optionStrategy, 1);
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm
LongAndShortCallCalendarSpreadStrategiesAlgorithm.cs C#

Option Strategies

Long Put Calendar Spread

Introduction

Put calendar spread , also known as put horizontal spread , is a combination of a longer-term (far-leg/front-month) put and a shorter-term (near-leg/back-month) put, where both puts have the same underlying stock and the same strike price. The long put calendar spread consists of buying a longer-term put and selling a shorter-term put. This strategy profits from a decrease in price movement. The strategy also profits from the time decay value because the theta \theta (the Option price decay by 1 day closer to maturity) of the shorter-term put is larger than the longer-term put.

Implementation

Follow these steps to implement the long put calendar spread strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(500000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().PutCalendarSpread(0, 30, 60));
}
```

   The `PutCalendarSpread` filter narrows the universe down to just the two contracts you need to form a long put calendar spread.

2. In the `OnData` method, select the strike price and expiration dates of the contracts in the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Get the ATM strike
    var atmStrike = chain.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First().Strike;

    // Select the ATM put Option contracts
    var puts = chain.Where(x => x.Strike == atmStrike && x.Right == OptionRight.Put);
    if (puts.Count() == 0) return;

    // Select the near and far expiry contracts
    var expiries = puts.Select(x => x.Expiry).ToList();
    var nearExpiry = expiries.Min();
    var farExpiry = expiries.Max();
```

3. In the `OnData` method, select the contracts and place the orders.

Approach A: Call the `OptionStrategies.PutCalendarSpread` method with the details of each leg and then pass the result to the `Buy` method.

```
C#
var optionStrategy = OptionStrategies.PutCalendarSpread(_symbol, atmStrike, nearExpiry, farExpiry);
Buy(optionStrategy, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

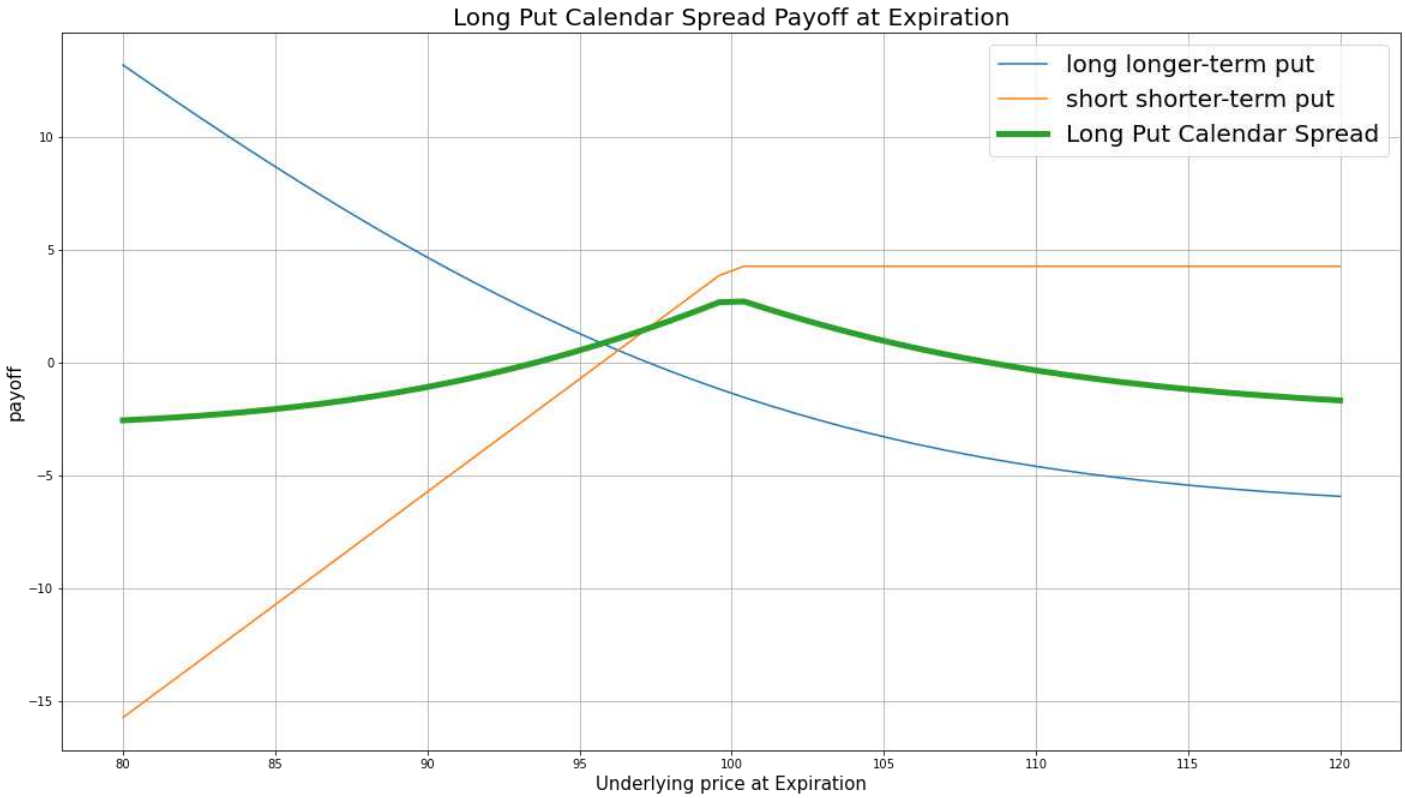
```
C#
var nearExpiryPut = puts.Single(x => x.Expiry == nearExpiry);
var farExpiryPut = puts.Single(x => x.Expiry == farExpiry);

var legs = new List<Leg>()
    {
        Leg.Create(nearExpiryPut.Symbol, -1),
        Leg.Create(farExpiryPut.Symbol, 1)
    };
ComboMarketOrder(legs, 1);
```

Strategy Payoff

The long put calendar spread is a limited-reward-limited-risk strategy. The payoff is taken at the shorter-term expiration. The payoff is

$$\begin{array}{rcll} P^{\textrm{short-term}}_T & = & (K - S_T)^{+}\\ P_T & = & (P^{\textrm{long-term}}_T - P^{\textrm{short-term}}_T + P^{\textrm{short-term}}_0 - P^{\textrm{long-term}}_0)\times m - fee \end{array} \begin{array}{rcl} \textrm{where} & P^{\textrm{short-term}}_T & = & \textrm{Shorter term put value at time T}\\ & P^{\textrm{long-term}}_T & = & \textrm{Longer term put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K & = & \textrm{Strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & P^{\textrm{short-term}}_0 & = & \textrm{Shorter term put value at position opening (credit received)}\\ & P^{\textrm{long-term}}_0 & = & \textrm{Longer term put value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of shorter term put expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is undetermined because it depends on the underlying volatility. It occurs when $S_T = S_0$ and the spread

of the puts are at their maximum.

The maximum loss is the net debit paid, $P^{\textrm{short-term}}_0 - P^{\textrm{long-term}}_0$. It occurs when the underlying price moves very deep ITM or OTM so the values of both puts are close to zero.

If the Option is American Option, there is a risk of early assignment on the contract you sell. Naked long puts pose risk of losing all the debit paid if you don't close the position with short put together and the price drops below its strike.

Example

The following table shows the price details of the assets in the long put calendar spread algorithm:

Asset	Price ($)	Strike ($)
Shorter-term put at position opening	11.30	800.00
Longer-term put at position opening	19.30	800.00
Longer-term put at shorter-term expiration	3.50	800.00
Underlying Equity at shorter-term expiration	828.07	-

Therefore, the payoff is

$$\begin{array}{rcll} P^{\textrm{short-term}}_T & = & (K - S_T)^{+}\\ & = & (800.00-828.07)^{+}\\ & = & 0\\ P_T & = & (P^{\textrm{long-term}}_T - P^{\textrm{short-term}}_T + P^{\textrm{short-term}}_0 - P^{\textrm{long-term}}_0)\times m - fee\\ & = & (3.50-0+11.30-19.30)\times100-1.00\times2\\ & = & -452\\ \end{array}$$

So, the strategy loses $452.

The following algorithm implements a long put calendar spread Option strategy:

```C#
public class OptionStrategy : QCAlgorithm
{
    private Symbol _equity;
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(500000);

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().PutCalendarSpread(0, 30, 60));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested) return;

        // Get the OptionChain of the symbol
        var chain = slice.OptionChains.get(_symbol, null);
        if (chain == null || chain.Count() == 0) return;

        // get at-the-money strike
        var atmStrike = chain.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First().Strike;

        // filter the put options from the contracts which is ATM in the option chain.
        var puts = chain.Where(x => x.Strike == atmStrike && x.Right == OptionRight.Put);
        if (puts.Count() == 0) return;

        // sorted the optionchain by expiration date
        var expiries = puts.Select(x => x.Expiry).OrderBy(x => x);

        // select the farest expiry as far-leg expiry, and the nearest expiry as near-leg expiry
        var nearExpiry = expiries.First();
        var farExpiry = expiries.Last();

        var optionStrategy = OptionStrategies.PutCalendarSpread(_symbol, atmStrike, nearExpiry, farExpiry);
        // We open a position with 1 unit of the option strategy
        Buy(optionStrategy, 1);    // if long put calendar spread
        //Sell(optionStrategy, 1);   // if short put calendar spread
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm
IndexOptionPutCalendarSpreadAlgorithm.cs C# LongAndShortPutCalendarSpreadStrategiesAlgorithm.cs C#

Option Strategies

Short Put Calendar Spread

Introduction

Put calendar spread , also known as put horizontal spread , is a combination of a longer-term (far-leg/front-month) put and a shorter-term (near-leg/back-month) put, where both puts have the same underlying stock and the same strike price. The short put calendar spread consists of selling a longer-term put and buying a shorter-term put. This strategy profits from an increase in price movement.

Implementation

Follow these steps to implement the short put calendar spread strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(500000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().PutCalendarSpread(0, 30, 60));
}
```

   The `PutCalendarSpread` filter narrows the universe down to just the two contracts you need to form a short put calendar spread.

2. In the `OnData` method, select the strike price and expiration dates of the contracts in the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Get the ATM strike
    var atmStrike = chain.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First().Strike;

    // Select the ATM put Option contracts
    var puts = chain.Where(x => x.Strike == atmStrike && x.Right == OptionRight.Put);
    if (puts.Count() == 0) return;

    // Select the near and far expiry contracts
    var expiries = puts.Select(x => x.Expiry).ToList();
    var nearExpiry = expiries.Min();
    var farExpiry = expiries.Max();
```

3. In the `OnData` method, select the contracts and place the orders.

Approach A: Call the `OptionStrategies.ShortPutCalendarSpread` method with the details of each leg and then pass the result to the `Buy` method.

```
C#
var optionStrategy = OptionStrategies.ShortPutCalendarSpread(_symbol, atmStrike, nearExpiry, farExpiry);
Buy(optionStrategy, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

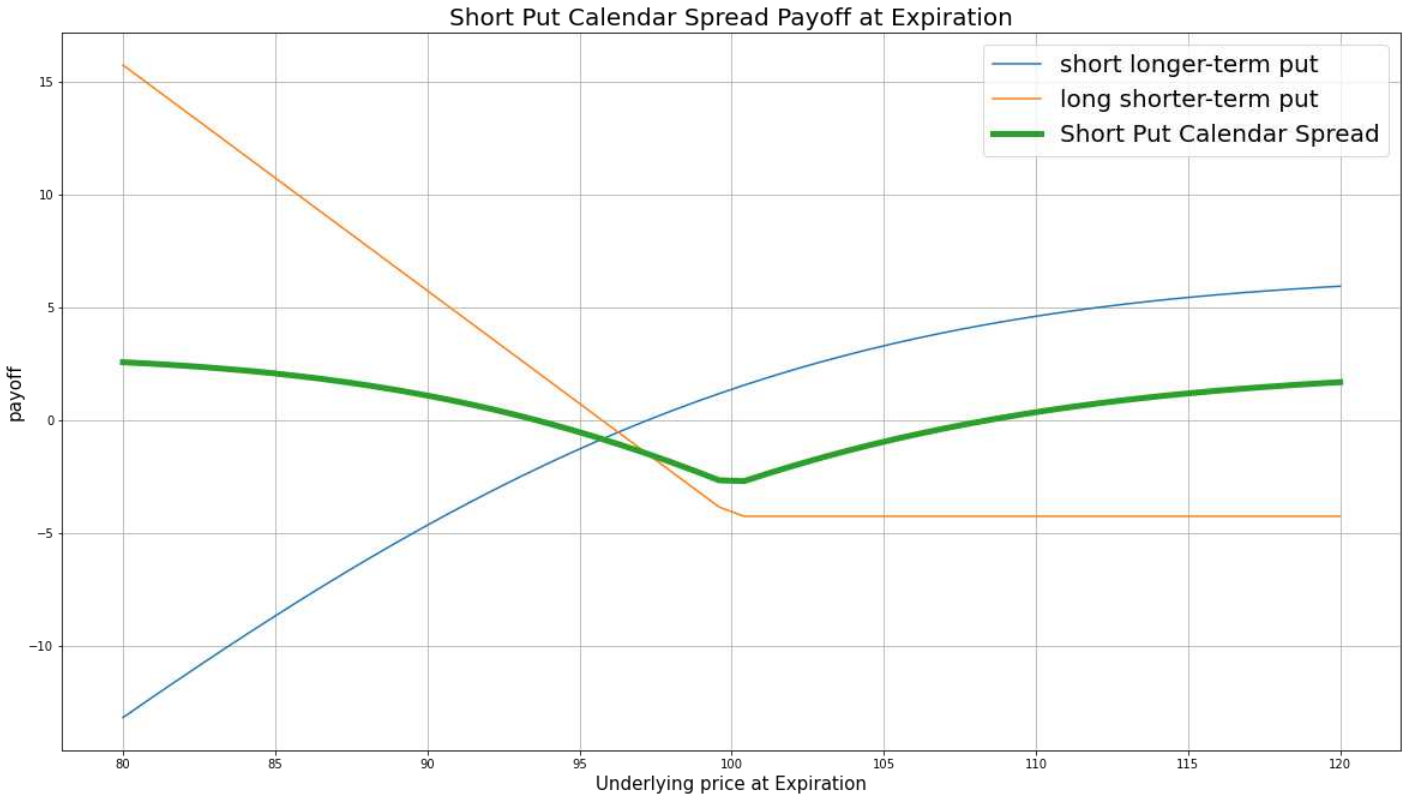
```
C#
var nearExpiryPut = puts.Single(x => x.Expiry == nearExpiry);
var farExpiryPut = puts.Single(x => x.Expiry == farExpiry);

var legs = new List<Leg>()
    {
        Leg.Create(nearExpiryPut.Symbol, 1),
        Leg.Create(farExpiryPut.Symbol, -1)
    };
ComboMarketOrder(legs, 1);
```

Strategy Payoff

The short put calendar spread is a limited-reward-limited-risk strategy. The payoff is taken at the shorter-term expiration. The payoff is

$$\begin{array}{rcll} P^{\textrm{short-term}}_T & = & (K - S_T)^{+}\\ P_T & = & (P^{\textrm{short-term}}_T - P^{\textrm{long-term}}_T + P^{\textrm{long-term}}_0 - P^{\textrm{short-term}}_0)\times m - fee \end{array} \begin{array}{rcl} \textrm{where} & & \\ P^{\textrm{short-term}}_T & = & \textrm{Shorter term put value at time T}\\ P^{\textrm{long-term}}_T & = & \textrm{Longer term put value at time T}\\ S_T & = & \textrm{Underlying asset price at time T}\\ K & = & \textrm{Strike price}\\ P_T & = & \textrm{Payout total at time T}\\ P^{\textrm{short-term}}_0 & = & \textrm{Shorter term put value at position opening (debit paid)}\\ P^{\textrm{long-term}}_0 & = & \textrm{Longer term put value at position opening (credit received)}\\ m & = & \textrm{Contract multiplier}\\ T & = & \textrm{Time of shorter term put expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is the net credit received, $P^{\textrm{long-term}}_0 - P^{\textrm{short-term}}_0$. It occurs when the

underlying price moves very deep ITM or OTM so the values of both puts are close to zero.

The maximum loss is undetermined because it depends on the underlying volatility. It occurs when $S_T = S_0$ and the spread of the 2 puts are at their maximum.

If the Option is American Option, there is a risk of early assignment on the contract you sell. Additionally, if you don't close the put positions together, the naked short put will have unlimited drawdown risk after the long put expires.

Example

The following table shows the price details of the assets in the short put calendar spread algorithm:

Asset	Price ($)	Strike ($)
Shorter-term put at position opening	11.30	800.00
Longer-term put at position opening	19.30	800.00
Longer-term put at shorter-term expiration	3.50	800.00
Underlying Equity at shorter-term expiration	828.07	-

Therefore, the payoff is

$$\begin{array}{rcll} P^{\textrm{short-term}}_T & = & (K - S_T)^{+}\\ & = & (800.00-828.07)^{+}\\ & = & 0\\ P_T & = & (-P^{\textrm{long-term}}_T + P^{\textrm{short-term}}_T - P^{\textrm{short-term}}_0 + P^{\textrm{long-term}}_0)\times m - fee\\ & = & (-3.50+0-11.30+19.30)\times100-1.00\times2\\ & = & 448\\ \end{array}$$

So, the strategy gains $448.

The following algorithm implements a short put calendar spread Option strategy:

C#

```
public class OptionStrategy : QCAlgorithm
{
    private Symbol _equity;
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(500000);

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().PutCalendarSpread(0, 30, 60));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested) return;

        // Get the OptionChain of the symbol
        var chain = slice.OptionChains.get(_symbol, null);
        if (chain == null || chain.Count() == 0) return;

        // get at-the-money strike
        var atmStrike = chain.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First().Strike;

        // filter the put options from the contracts which is ATM in the option chain.
        var puts = chain.Where(x => x.Strike == atmStrike && x.Right == OptionRight.Put);
        if (puts.Count() == 0) return;

        // sorted the optionchain by expiration date
        var expiries = puts.Select(x => x.Expiry).OrderBy(x => x);

        // select the farest expiry as far-leg expiry, and the nearest expiry as near-leg expiry
        var nearExpiry = expiries.First();
        var farExpiry = expiries.Last();

        var optionStrategy = OptionStrategies.ShortPutCalendarSpread(_symbol, atmStrike, nearExpiry, farExpiry);
        // We open a position with 1 unit of the option strategy
        Buy(optionStrategy, 1);
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm
LongAndShortPutCalendarSpreadStrategiesAlgorithm.cs C#

Option Strategies

Covered Call

Introduction

A Covered Call consists of a long position in a stock and a short position in call Options for the same amount of stock. Covered calls aim to profit from the Option premium by selling calls written on the stock you already own. At any time for American Options or at expiration for European Options, if the stock moves below the strike price, you keep the premium and still maintain the underlying Equity position. If the underlying price moves above the strike, the Option buyer can exercise the Options contract , which means you sell your stock at the strike price and keep the premium. Another risk of a covered call comes from the long stock position, which can drop in value.

Implementation

Follow these steps to implement the covered call strategy:

1. In the `Initialize` method, set the start date, end date, starting cash, and Options universe .

```
C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(100000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("IBM");
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().NakedCall(30, 0));
}
```

   The `NakedCall` filter narrows the universe down to just the one contract you need to form a covered call.

2. In the `OnData` method, select the Option contract.

```
C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Find ATM call with the farthest expiry
    var expiry = chain.Max(x => x.Expiry);
    var atmCall = chain
        .Where(x => x.Right == OptionRight.Call && x.Expiry == expiry)
        .OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price))
        .FirstOrDefault();
```

3. In the `OnData` method, place the orders.

   Approach A: Call the `OptionStrategies.CoveredCall` method with the details of each leg and then pass the result to the `Buy` method.

   ```C#
   var coveredCall = OptionStrategies.CoveredCall(_symbol, atmCall.Strike, expiry);
   Buy(coveredCall, 1);
   ```

   Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

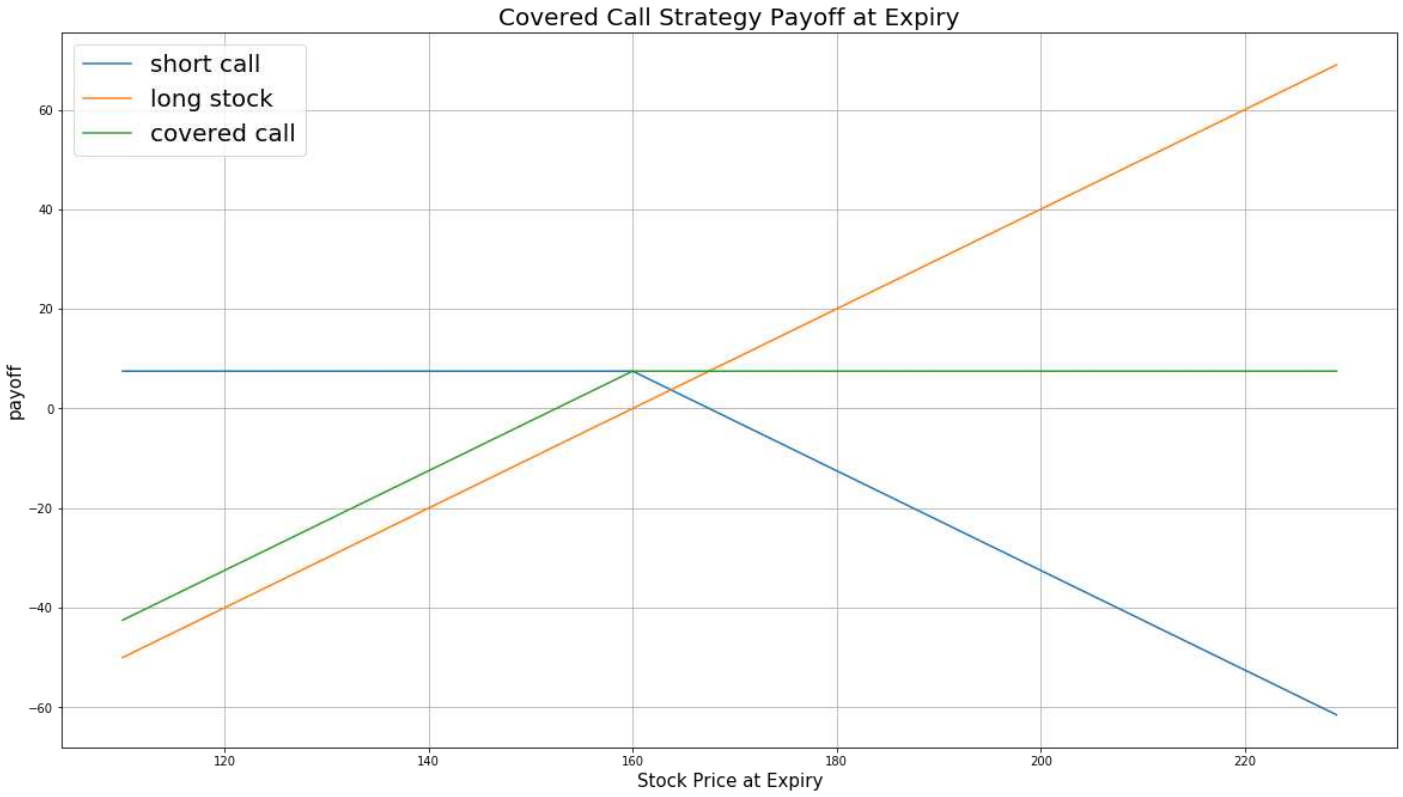
   ```C#
   var legs = new List<Leg>()
       {
           Leg.Create(atmCall.Symbol, -1),
           Leg.Create(chain.Underlying.Symbol, chain.Underlying.SymbolProperties.ContractMultiplier)
       };
   ComboMarketOrder(legs, 1);
   ```

Strategy Payoff

The payoff of the strategy is

$$\begin{array}{rcll} C^{K}_T & = & (S_T - K)^{+}\\ P_T & = & (S_T - S_0 + C^{K}_0 - C^{K}_T)\times m - fee \end{array}$$
$$\begin{array}{rcll} \textrm{where} & C^{K}_T & = & \textrm{Call value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K & = & \textrm{Call strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & S_0 & = & \textrm{Underlying asset price when the trade opened}\\ & C^{K}_0 & = & \textrm{Call price when the trade opened (credit received)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is $K - S_T + C^{K}_0$, which occurs when the underlying price is at or above the strike price of the call at expiration.

The maximum loss is $S_0 - C^{K}_0$, which ocurrs when the underlying price drops.

If the Option is American Option, there is a risk of early assignment on the contract you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	Price ($)	Strike ($)
Call	3.35	185.00
Underlying Equity at start of the trade	187.07	-
Underlying Equity at expiration	190.01	-

Therefore, the payoff is

$$\begin{array}{rcll} C^{K}_T & = & (S_T - K)^{+}\\ & = & (190.01 - 185)^{+}\\ & = & 5.01\\ P_T & = & (S_T - S_0 + C^{K}_0 - C^{K}_T)\times m - fee\\ & = & (190.01 - 187.07 + 3.35 - 5.01)\times m - fee\\ & = & 1.28 \times 100 - 2\\ & = & 126 \end{array}$$

So, the strategy gains $126.

The following algorithm implements a covered call strategy:

```C#
public class CoveredCallAlgorithm : QCAlgorithm
{
    private Symbol _call, _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("IBM");
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().NakedCall(30, 0));

        // use the underlying equity as the benchmark
        SetBenchmark(_symbol.Underlying);
    }

    public override void OnData(Slice slice)
    {
        if (_call != null && Portfolio[_call].Invested) return;

        if (!slice.OptionChains.TryGetValue(_symbol, out var chain)) return;

        // Find ATM call with the farthest expiry
        var expiry = chain.Max(x => x.Expiry);
        var atmCall = chain
            .Where(x=> x.Right == OptionRight.Call && x.Expiry == expiry)
            .OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price))
            .FirstOrDefault();

        if (atmCall == null) return;

        var coveredCall = OptionStrategies.CoveredCall(_symbol, atmCall.Strike, expiry);
        Buy(coveredCall, 1);

        _call = atmCall.Symbol;
    }
}
```

Option Strategies

Covered Put

Introduction

A Covered Put consists of a short position in a stock and a short position in put Options for the same amount of stock. Covered puts aim to profit from the Option premium by selling puts written on the stock you already shorted. At any time for American Options or at expiration for European Options, if the stock moves below the strike price, you keep the premium and still maintain the underlying Equity position. If the underlying price moves above the strike, the Option buyer can exercise the Options contract , which mean you buy the stock at the strike price but you will still keep the premium. Another risk of a covered put comes from the short stock position, which can drop in value.

Implementation

Follow these steps to implement the covered put strategy:

1. In the `Initialize` method, set the start date, end date, starting cash, and Options universe .

```
C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(100000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("IBM");
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().NakedPut(30, 0));
}
```

   The `NakedPut` filter narrows the universe down to just the one contract you need to form a covered put.

2. In the `OnData` method, select the Option contract.

```
C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Find ATM put with the farthest expiry
    var expiry = chain.Max(x => x.Expiry);
    var atmput = chain
        .Where(x => x.Right == OptionRight.Put && x.Expiry == expiry)
        .OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price))
        .FirstOrDefault();
```

3. In the `OnData` method, place the orders.

   Approach A: Call the `OptionStrategies.CoveredPut` method with the details of each leg and then pass the result to the `Buy` method.

```C#
var coveredPut = OptionStrategies.CoveredPut(_symbol, atmput.Strike, expiry);
Buy(coveredPut, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

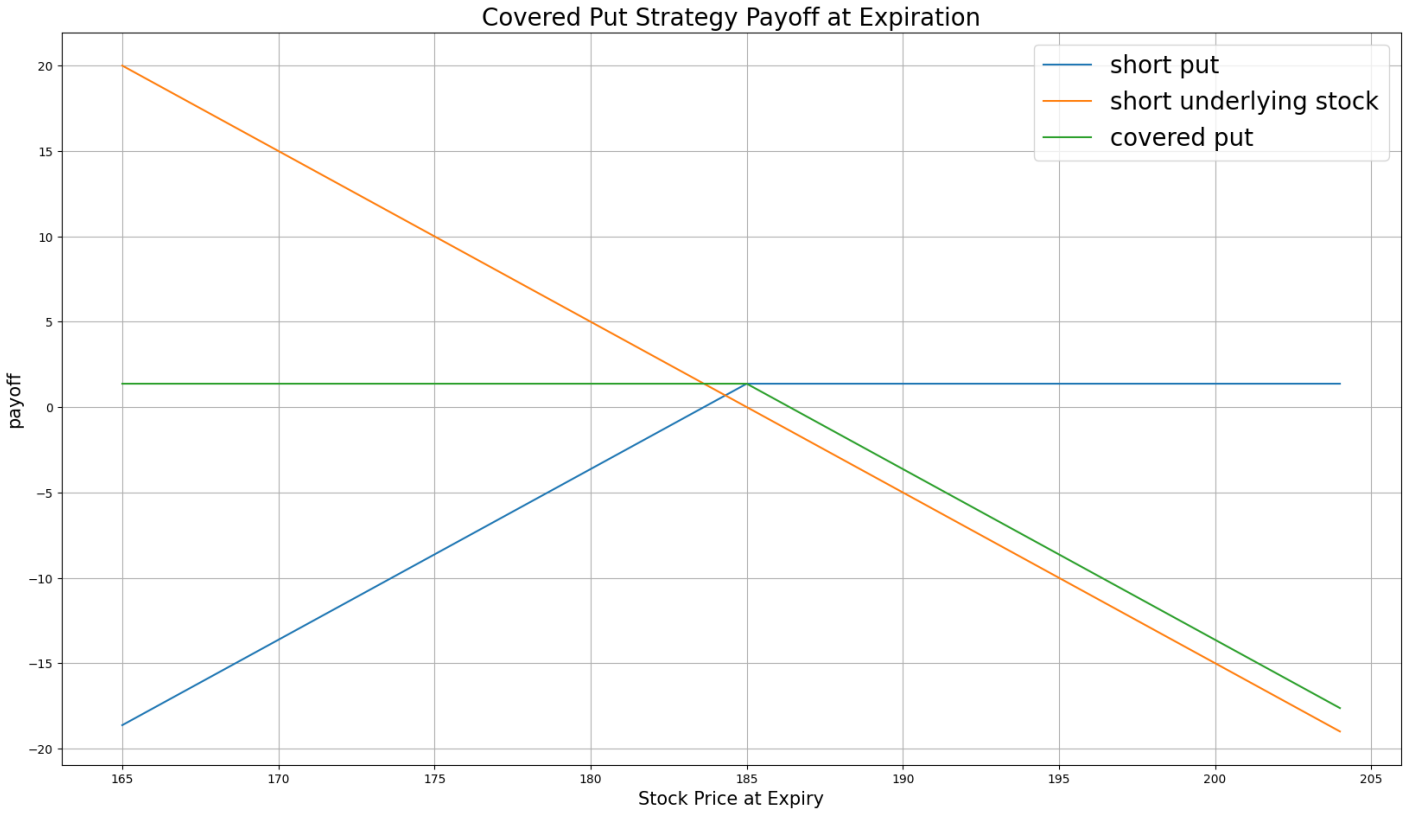
```C#
var legs = new List<Leg>()
    {
        Leg.Create(atmPut.Symbol, -1),
        Leg.Create(chain.Underlying.Symbol, -chain.Underlying.SymbolProperties.ContractMultiplier)
    };
ComboMarketOrder(legs, 1);
```

Strategy Payoff

The payoff of the strategy is

$$\begin{array}{rcl} P^{K}_T & = & (K - S_T)^{+}\\ P_T & = & (S_0 - S_T + P^{K}_0 - P^{K}_T)\times m - fee \end{array}$$
$$\begin{array}{rcl} \textrm{where} & P^{K}_T & = & \textrm{Put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K & = & \textrm{Put strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & S_0 & = & \textrm{Underlying asset price when the trade opened}\\ & P^{K}_0 & = & \textrm{Put price when the trade opened (credit received)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is $S_T - K + P^{K}_0$. It occurs when the underlying price is at or below the strike price of the put at expiration.

If the underlying price increase, the maximum loss is unlimited.

If the Option is American Option, there is a risk of early assignment on the contract you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	Price ($)	Strike ($)
Put	1.37	185.00
Underlying Equity at start of the trade	186.94	-
Underlying Equity at expiration	190.01	-

Therefore, the payoff is

$$\begin{array}{rcll} P^{K}_T & = & (K - S_T)^{+}\\ & = & (185 - 190.01)^{+}\\ & = & 0\\ P_T & = & (S_0 - S_T + P^{K}_0 - P^{K}_T)\times m - fee\\ & = & (186.94 - 190.01 + 1.37 - 0)\times m - fee\\ & = & -1.70 \times 100 - 2\\ & = & -172 \end{array}$$

So, the strategy loses $172.

The following algorithm implements a covered put strategy:

```C#
public class CoveredPutAlgorithm : QCAlgorithm
{
    private Symbol _put, _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("IBM");
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().NakedPut(30, 0));

        // use the underlying equity as the benchmark
        SetBenchmark(_symbol.Underlying);
    }

    public override void OnData(Slice slice)
    {
        if (_put != null && Portfolio[_put].Invested) return;

        if (!slice.OptionChains.TryGetValue(_symbol, out var chain)) return;

        // Find ATM put with the farthest expiry
        var expiry = chain.Max(x => x.Expiry);
        var atmPut = chain
            .Where(x=> x.Right == OptionRight.Put && x.Expiry == expiry)
            .OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price))
            .FirstOrDefault();

        if (atmPut == null) return;

        var coveredPut = OptionStrategies.CoveredPut(_symbol, atmPut.Strike, expiry);
        Buy(coveredPut, 1);

        _put = atmPut.Symbol;
    }
}
```

Option Strategies

Long Iron Butterfly

Introduction

The Long Iron Butterfly is an option strategy which involves four Option contracts. All the contracts have the same underlying stock and expiration, but the order of strike prices for the four contracts is A>B>C. The following table describes the strike price of each contract:

Position	Strike
1 OTM call	A
-1 ATM call	B
-1 ATM put	B
1 OTM put	C=B-(A-B)

The long iron butterfly consists of selling an OTM call, selling an OTM put, buying an ATM call, and buying an ATM put. This strategy profits from a decrease in price movement (implied volatility).

Implementation

Follow these steps to implement the long iron butterfly strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
C#
```

```
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(100000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().IronButterfly(30, 5));
}
```

   The `IronButterfly` filter narrows the universe down to just the four contracts you need to form a long iron butterfly.

2. In the `OnData` method, select the contracts in the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select expiry
    var expiry = chain.Max(x => x.Expiry);

    // Separate the call and put contracts
    var calls = chain.Where(x => x.Right == OptionRight.Call  && x.Expiry == expiry);
    var puts = chain.Where(x => x.Right == OptionRight.Put && x.Expiry == expiry);
    if (calls.Count() == 0 || puts.Count() == 0) return;

    // Get the ATM and OTM strike prices
    var atmStrike = calls.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First().Strike;
    var otmPutStrike = puts.Min(x => x.Strike);
    var otmCallStrike = 2 * atmStrike - otmPutStrike;
```

3. In the `OnData` method, select the contracts and place the orders.

   Approach A: Call the `OptionStrategies.IronButterfly` method with the details of each leg and then pass the result to the `Buy` method.

   ```C#
   var ironButterfly = OptionStrategies.IronButterfly(_symbol, otmPutStrike, atmStrike, otmCallStrike, expiry);
   Buy(ironButterfly, 2);
   ```

   Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

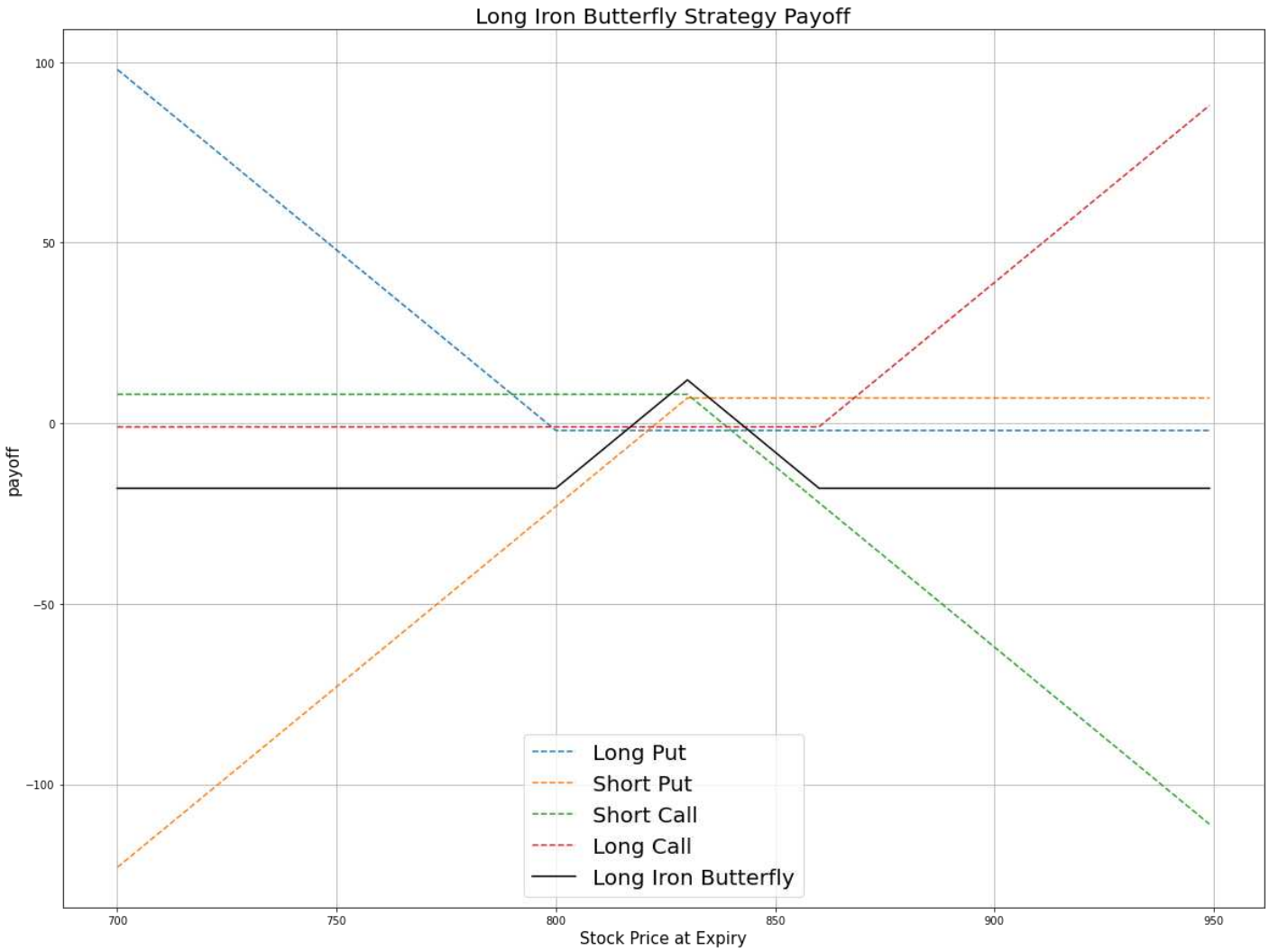
   ```C#
   // Select the contracts
   var atmCall = calls.Single(x => x.Strike == atmStrike);
   var atmPut = puts.Single(x => x.Strike == atmStrike);
   var otmCall = calls.Single(x => x.Strike == otmCallStrike);
   var otmPut = puts.Single(x => x.Strike == otmPutStrike);

   var legs = new List<Leg>()
       {
           Leg.Create(atmCall.Symbol, -1),
           Leg.Create(atmPut.Symbol, -1),
           Leg.Create(otmCall.Symbol, 1),
           Leg.Create(otmPut.Symbol, 1)
       };
   ComboMarketOrder(legs, 1);
   ```

Strategy Payoff

The long call butterfly is a limited-reward-limited-risk strategy. The payoff is

$$\begin{array}{rcll} C^{OTM}_T & = & (S_T - K^C_{OTM})^{+}\\ C^{ATM}_T & = & (S_T - K^C_{ATM})^{+}\\ P^{OTM}_T & = & (K^P_{OTM} - S_T)^{+}\\ P^{ATM}_T & = & (K^P_{ATM} - S_T)^{+}\\ P_T & = & (C^{OTM}_T + P^{OTM}_T - C^{ATM}_T - P^{ATM}_T - C^{OTM}_0 - P^{OTM}_0 + C^{ATM}_0 + P^{ATM}_0)\times m - fee \end{array} \begin{array}{rcll} \textrm{where} & C^{OTM}_T & = & \textrm{OTM call value at time T}\\ & C^{ATM}_T & = & \textrm{ATM call value at time T}\\ & P^{OTM}_T & = & \textrm{OTM put value at time T}\\ & P^{ATM}_T & = & \textrm{ATM put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^C_{OTM} & = & \textrm{OTM call strike price}\\ & K^C_{ATM} & = & \textrm{ATM call strike price}\\ & K^P_{OTM} & = & \textrm{OTM put strike price}\\ & K^P_{ATM} & = & \textrm{ATM put strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & C^{OTM}_0 & = & \textrm{OTM call value at position opening (debit paid)}\\ & C^{ATM}_0 & = & \textrm{ATM call value at position opening (credit received)}\\ & P^{OTM}_0 & = & \textrm{OTM put value at position opening (debit paid)}\\ & P^{ATM}_0 & = & \textrm{ATM put value at position opening (credit received)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is the net credit received, C^{ATM}_0 + P^{ATM}_0 - C^{OTM}_0 - P^{OTM}_0. It occurs when the underlying price stays the same as when you opened the trade.

The maximum loss is K^C_{OTM} - K^C_{ATM} - C^{ATM}_0 - P^{ATM}_0 + C^{OTM}_0 + P^{OTM}_0. It occurs when the underlying price is below the OTM put strike price or above the OTM call strike price at expiration.

If the Option is American Option, there is a risk of early assignment on the contracts you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	Price ($)	Strike ($)
OTM call	1.35	855.00
OTM put	1.50	810.00
ATM call	10.30	832.50
ATM put	9.50	832.50
Underlying Equity at expiration	843.25	-

Therefore, the payoff is

\begin{array}{rcll} C^{OTM}_T & = & (S_T - K^C_{OTM})^{+}\\ & = & (843.25-855.00)^{+}\\ & = & 0\\ C^{ATM}_T & = & (S_T - K^C_{ATM})^{+}\\ & = & (843.25-832.50)^{+}\\ & = & 10.75\\ P^{OTM}_T & = & (K^P_{OTM} - S_T)^{+}\\ & = & (810.00-843.25)^{+}\\ & = & 0\\ P^{ATM}_T & = & (K^P_{ATM} - S_T)^{+}\\ & = & (832.50.00-843.25)^{+}\\ & = & 0\\ P_T & = & (C^{ATM}_T + P^{ATM}_T - C^{OTM}_T - P^{OTM}_T - C^{ATM}_0 - P^{ATM}_0 + C^{OTM}_0 + P^{OTM}_0)\times m - fee\\ & = & (10.75+0-0-0-10.30-9.50+1.35+1.50)\times100-1\times4\\ & = & -624 \end{array}

So, the strategy losses $624.

The following algorithm implements a long iron butterfly Option strategy:

```C#
public class LongButterflyStrategy : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().IronButterfly(30, 5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            return;
        }

        // Select expiry
        var expiry = chain.Max(x => x.Expiry);

        // Separate the call and put contracts
        var calls = chain.Where(x => x.Right == OptionRight.Call  && x.Expiry == expiry);
        var puts = chain.Where(x => x.Right == OptionRight.Put && x.Expiry == expiry);
        if (calls.Count() == 0 || puts.Count() == 0) return;

        // Get the ATM and OTM strike prices
        var atmStrike = calls.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First().Strike;
        var otmPutStrike = puts.Min(x => x.Strike);
        var otmCallStrike = 2 * atmStrike - otmPutStrike;

        // Order Strategy
        var ironButterfly = OptionStrategies.IronButterfly(_symbol, otmPutStrike, atmStrike, otmCallStrike, expiry);
        Buy(ironButterfly, 1);
    }
}
```

Option Strategies

Short Iron Butterfly

Introduction

The Short Iron Butterfly is an option strategy which involves four Option contracts. All the contracts have the same underlying stock and expiration, but the order of strike prices for the four contracts is A>B>C. The following table describes the strike price of each contract:

Position	Strike
1 OTM call	A
-1 ATM call	B
-1 ATM put	B
1 OTM put	C=B-(A-B)

The short call butterfly consists of buying an OTM call, buying an OTM put, selling an ATM call, and selling an ATM put. This strategy profits from an increase in price movement (implied volatility) and from time decay value since ATM options decay sharper.

Implementation

Follow these steps to implement the short iron butterfly strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(100000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().IronButterfly(30, 5));
}
```

   The `IronButterfly` filter narrows the universe down to just the four contracts you need to form a short iron butterly.

2. In the `OnData` method, select the contracts in the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select expiry
    var expiry = chain.Max(x => x.Expiry);

    // Separate the call and put contracts
    var calls = chain.Where(x => x.Right == OptionRight.Call  && x.Expiry == expiry);
    var puts = chain.Where(x => x.Right == OptionRight.Put && x.Expiry == expiry);
    if (calls.Count() == 0 || puts.Count() == 0) return;

    // Get the ATM and OTM strike prices
    var atmStrike = calls.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First().Strike;
    var otmPutStrike = puts.Min(x => x.Strike);
    var otmCallStrike = 2 * atmStrike - otmPutStrike;
```

3. In the `OnData` method, select the contracts and place the orders.

   Approach A: Call the `OptionStrategies.ShortIronButterfly` method with the details of each leg and then pass the result to the `Buy` method.

   ```C#
   var shortIronButterfly = OptionStrategies.ShortIronButterfly(_symbol, otmPutStrike, atmStrike, otmCallStrike,
   expiry);
   Buy(shortIronButterfly, 2);
   ```

   Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

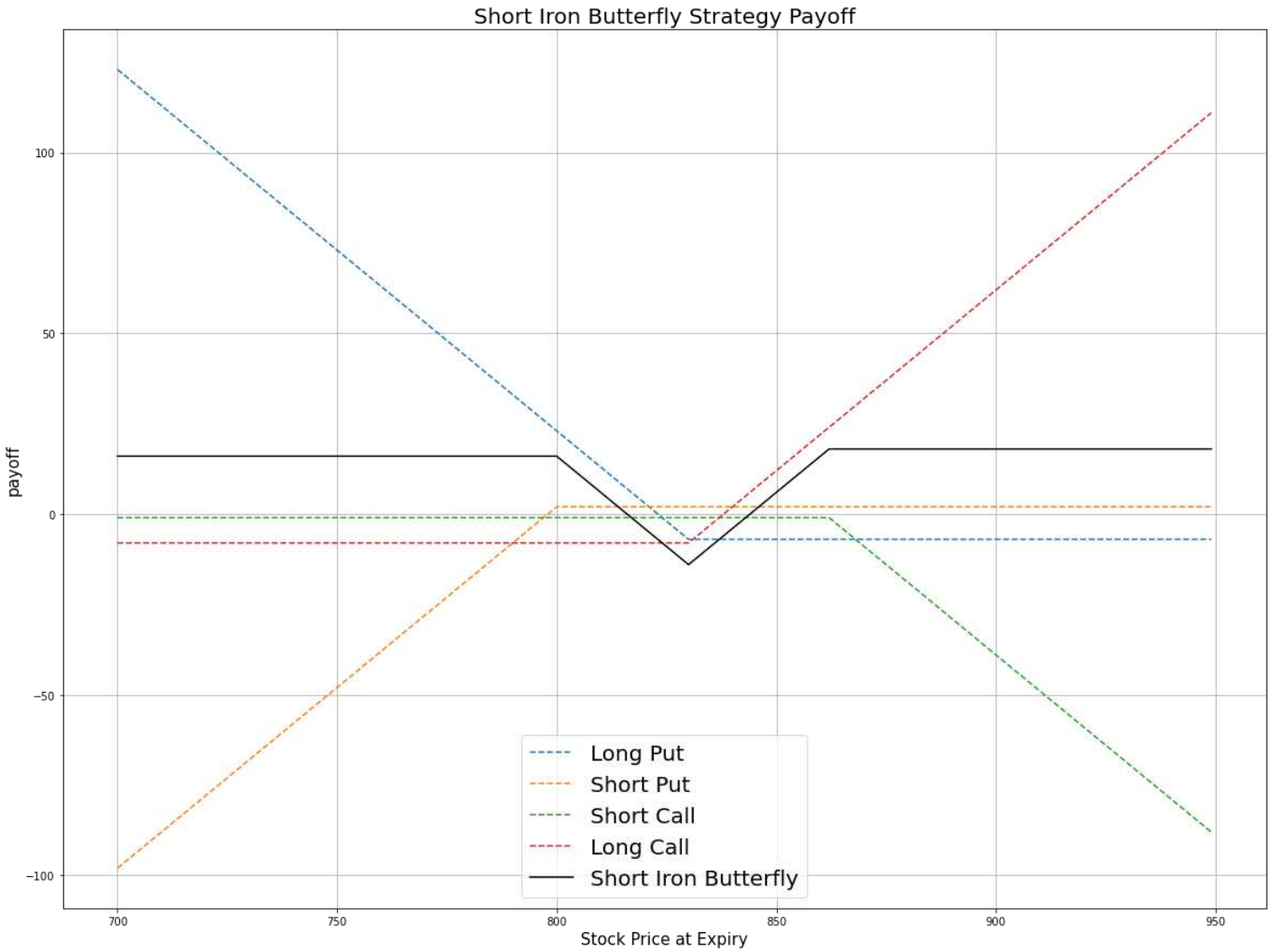
   ```C#
   // Select the contracts
   var atmCall = calls.Single(x => x.Strike == atmStrike);
   var atmPut = puts.Single(x => x.Strike == atmStrike);
   var otmCall = calls.Single(x => x.Strike == otmCallStrike);
   var otmPut = puts.Single(x => x.Strike == otmPutStrike);

   var legs = new List<Leg>()
       {
           Leg.Create(atmCall.Symbol, 1),
           Leg.Create(atmPut.Symbol, 1),
           Leg.Create(otmCall.Symbol, -1),
           Leg.Create(otmPut.Symbol, -1)
       };
   ComboMarketOrder(legs, 1);
   ```

Strategy Payoff

The short iron butterfly is a limited-reward-limited-risk strategy. The payoff is

$$\begin{array}{rcll} C^{OTM}_T & = & (S_T - K^C_{OTM})^{+}\\ C^{ATM}_T & = & (S_T - K^C_{ATM})^{+}\\ P^{OTM}_T & = & (K^P_{OTM} - S_T)^{+}\\ P^{ATM}_T & = & (K^P_{ATM} - S_T)^{+}\\ P_T & = & (C^{ATM}_T + P^{ATM}_T - C^{OTM}_T - P^{OTM}_T - C^{ATM}_0 - P^{ATM}_0 + C^{OTM}_0 + P^{OTM}_0)\times m - fee \end{array} \begin{array}{rcll} \textrm{where} & C^{OTM}_T & = & \textrm{OTM call value at time T}\\ & C^{ATM}_T & = & \textrm{ATM call value at time T}\\ & P^{OTM}_T & = & \textrm{OTM put value at time T}\\ & P^{ATM}_T & = & \textrm{ATM put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^C_{OTM} & = & \textrm{OTM call strike price}\\ & K^C_{ATM} & = & \textrm{ATM call strike price}\\ & K^P_{OTM} & = & \textrm{OTM put strike price}\\ & K^P_{ATM} & = & \textrm{ATM put strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & C^{OTM}_0 & = & \textrm{OTM call value at position opening (credit received)}\\ & C^{ATM}_0 & = & \textrm{ATM call value at position opening (debit paid)}\\ & P^{OTM}_0 & = & \textrm{OTM put value at position opening (credit received)}\\ & P^{ATM}_0 & = & \textrm{ATM put value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is K^C_{OTM} - K^C_{ATM} - C^{ATM}_0 - P^{ATM}_0 + C^{OTM}_0 + P^{OTM}_0. It occurs when the underlying price is below the OTM put strike price or above the OTM call strike price at expiration.

The maximum loss is the net debit paid, C^{OTM}_0 + P^{OTM}_0 - C^{ATM}_0 - P^{ATM}_0. It occurs when the underlying price stays the same as when you opened the trade.

If the Option is American Option, there is a risk of early assignment on the contracts you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	**Price ($)**	**Strike ($)**
OTM call	2.35	855.00
OTM put	2.75	810.00
ATM call	8.10	832.50
ATM put	7.40	832.50
Underlying Equity at expiration	843.25	-

Therefore, the payoff is

\begin{array}{rcll} C^{OTM}_T & = & (S_T - K^C_{OTM})^{+}\\ & = & (843.25-855.00)^{+}\\ & = & 0\\ C^{ATM}_T & = & (S_T - K^C_{ATM})^{+}\\ & = & (843.25-832.50)^{+}\\ & = & 10.75\\ P^{OTM}_T & = & (K^P_{OTM} - S_T)^{+}\\ & = & (810.00-843.25)^{+}\\ & = & 0\\ P^{ATM}_T & = & (K^P_{ATM} - S_T)^{+}\\ & = & (832.50.00-843.25)^{+}\\ & = & 0\\ P_T & = & (C^{OTM}_T + P^{OTM}_T - C^{ATM}_T - P^{ATM}_T - C^{OTM}_0 - P^{OTM}_0 + C^{ATM}_0 + P^{ATM}_0)\times m - fee\\ & = & (0+0-10.75-0-2.35-2.75+8.10+7.40)\times100-1\times4\\ & = & -39 \end{array}

So, the strategy losses $39.

The following algorithm implements a short iron butterfly Option strategy:

C#

```
public class ShortButterflyStrategy : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().IronButterfly(30, 5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            return;
        }

        // Select expiry
        var expiry = chain.Max(x => x.Expiry);

        // Separate the call and put contracts
        var calls = chain.Where(x => x.Right == OptionRight.Call  && x.Expiry == expiry);
        var puts = chain.Where(x => x.Right == OptionRight.Put && x.Expiry == expiry);
        if (calls.Count() == 0 || puts.Count() == 0) return;

        // Get the ATM and OTM strike prices
        var atmStrike = calls.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First().Strike;
        var otmPutStrike = puts.Min(x => x.Strike);
        var otmCallStrike = 2 * atmStrike - otmPutStrike;

        // Order Strategy
        var shortIronButterfly = OptionStrategies.ShortIronButterfly(_symbol, otmPutStrike, atmStrike, otmCallStrike,
expiry);
        Buy(shortIronButterfly, 1);
    }
}
```

Option Strategies

Long Iron Condor

Introduction

The Long Iron Condor is an Option strategy that consists of four contracts. All the contracts have the same underlying Equity and expiration, but the order of strike prices is A>B>C>D. The following table describes the strike prices of each contract:

Position	Strike
1 far-OTM call	A
-1 near-OTM call	B, where B > underlying\ price
-1 near-OTM put	C, where C < underlying\ price
1 far-OTM put	D, where C-D = A-B

The long iron condor consists of buying a far OTM call, buying a far OTM put, selling a near ATM call, and selling a near ATM put. This strategy profits from an decrease in price movement (implied volatility) and time decay since ATM options decay sharper.

Implementation

Follow these steps to implement the long iron condor strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(500000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG");
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().IronCondor(30, 5, 10));
}
```

   The `IronCondor` filter narrows the universe down to just the four contracts you need to form a long iron condor.

2. In the `OnData` method, select the contracts in the strategy legs.

```
C#
public override void OnData(Slice slice)
{
    if (Portfolio[_symbol.Underlying].Invested)
    {
        Liquidate();
    }

    if (Portfolio.Invested || !IsMarketOpen(_symbol) ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Find put and call contracts with the farthest expiry
    var expiry = chain.Max(x => x.Expiry);
    var contracts = chain.Where(x => x.Expiry == expiry).OrderBy(x => x.Strike);

    var putContracts = contracts.Where(x => x.Right == OptionRight.Put).ToArray();
    var callContracts = contracts.Where(x => x.Right == OptionRight.Call).ToArray();

    if (putContracts.Length < 2 || callContracts.Length < 2) return;

    // Select the strategy legs
    var nearCall = callContracts[0];
    var farCall = callContracts[1];
    var nearPut = putContracts[1];
    var farPut = putContracts.Single(x => x.Strike == nearPut.Strike - farCall.Strike + nearCall.Strike);
```

3. In the `OnData` method, place the orders.

   Approach A: Call the `OptionStrategies.IronCondor` method with the details of each leg and then pass the result to the `Buy` method.

```
C#
var ironCondor = OptionStrategies.IronCondor(
    _symbol,
    farPut.Strike,
    nearPut.Strike,
    nearCall.Strike,
    farCall.Strike,
    expiry);

Buy(ironCondor, 2);
```

   Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

```
C#
var legs = new List<Leg>()
    {
        Leg.Create(farPut.Symbol, 1),
        Leg.Create(nearPut.Symbol, -1),
        Leg.Create(farCall.Symbol, 1),
        Leg.Create(nearCall.Symbol, -1)
    };
ComboMarketOrder(legs, 1);
```

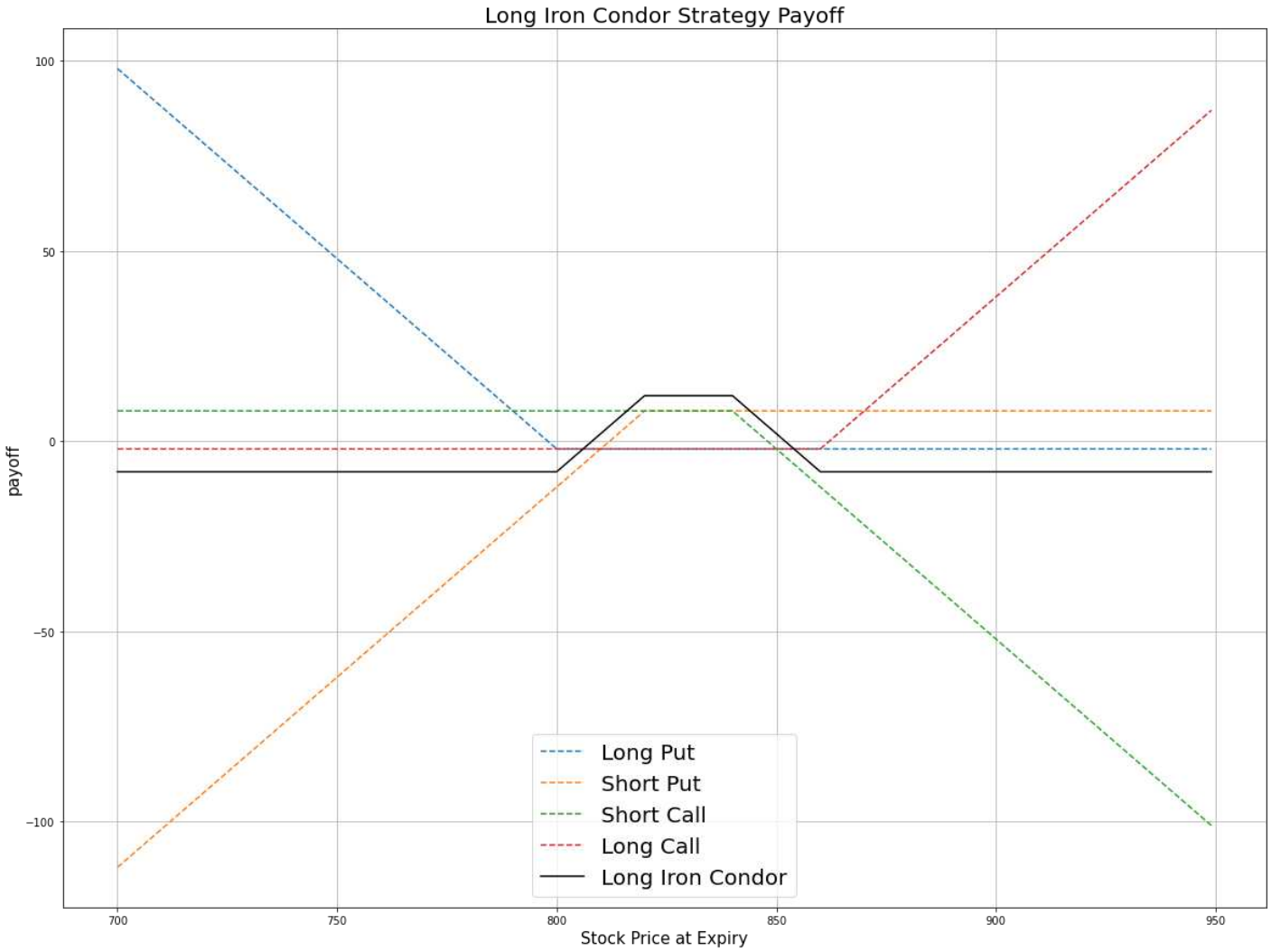
Strategy Payoff

This is a limited-reward-limited-risk strategy. The payoff is

$$\begin{array}{rcll} C^{far}_T & = & (S_T - K^C_{far})^{+}\\ C^{near}_T & = & (S_T - K^C_{near})^{+}\\ P^{far}_T & = & (K^P_{far} - S_T)^{+}\\ P^{near}_T & = & (K^P_{near} - S_T)^{+}\\ P_T & = & (C^{far}_T + P^{far}_T - C^{near}_T - P^{near}_T - C^{far}_0 - P^{far}_0 + C^{near}_0 + P^{near}_0)\times m - fee \end{array} \begin{array}{rcl} \textrm{where} & C^{far}_T & = & \textrm{Far OTM call value at time T}\\ & C^{near}_T & = & \textrm{Near OTM call value at time T}\\ & P^{far}_T & = & \textrm{Far OTM put value at time T}\\ & P^{near}_T & = & \textrm{Near ATM put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^C_{far} & = & \textrm{Far OTM call strike price}\\ & K^C_{near} & = & \textrm{Near OTM call strike price}\\ & K^P_{far} &$$

= & \textrm{Far OTM put strike price}\\ & K^P_{near} & = & \textrm{Near OTM put strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & C^{far}_0 & = & \textrm{Far OTM call value at position opening (credit received)}\\ & C^{near}_0 & = & \textrm{Near OTM call value at position opening (debit paid)}\\ & P^{far}_0 & = & \textrm{Far OTM put value at position opening (credit received)}\\ & P^{near}_0 & = & \textrm{Near OTM put value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}

The following chart shows the payoff at expiration:



The maximum profit is the net credit received after commission when opening the trade, where K^P_{OTM} < S_T < K^C_{OTM}.

The maximum loss is K^C_{far} - K^C_{near} + C^{near}_0 + P^{near}_0 - C^{far}_0 - P^{far}_0, where K^P_{OTM} > S_T or S_T > K^C_{OTM}.

If the Option is American Option, there is a risk of early assignment on the contracts you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	Price ($)	Strike ($)
Far-OTM call	1.85	857.50
Far-OTM put	3.80	815.00
Near-OTM call	1.65	852.50
Near-OTM put	3.50	820.00
Underlying Equity at expiration	843.25	-

Therefore, the payoff is

\begin{array}{rcll} C^{far}_T & = & (S_T - K^C_{far})^{+}\\ & = & (843.25-857.50)^{+}\\ & = & 0\\ C^{near}_T & = & (S_T - K^C_{near})^{+}\\ & = & (843.25-852.50)^{+}\\ & = & 0\\ P^{far}_T & = & (K^P_{far} - S_T)^{+}\\ & = & (815.00-843.25)^{+}\\ & = & 0\\ P^{near}_T & = & (K^P_{near} - S_T)^{+}\\ & = & (820.00-843.25)^{+}\\ & = & 0\\ P_T & = & (C^{near}_T + P^{near}_T - C^{far}_T - P^{far}_T - C^{near}_0 - P^{near}_0 + C^{far}_0 + P^{far}_0)\times m - fee\\ & = & (0+0-0-0+1.65+3.50-1.85-3.80)\times100-1\times4\\ & = & -54 \end{array}

So, the strategy loses $54.

The following algorithm implements a long iron condor Option strategy:

C#

```
public class LongIronCondorStrategy : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().IronCondor(30, 5, 10));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested || !IsMarketOpen(_symbol) ||
            !slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            return;
        }

        // Find put and call contracts with the farthest expiry
        var expiry = chain.Max(x => x.Expiry);
        var contracts = chain.Where(x => x.Expiry == expiry).OrderBy(x => x.Strike);

        var putContracts = contracts.Where(x => x.Right == OptionRight.Put).ToArray();
        var callContracts = contracts.Where(x => x.Right == OptionRight.Call).ToArray();

        if (putContracts.Length < 2 || callContracts.Length < 2) return;

        // Select the strategy legs
        var nearCall = callContracts[0];
        var farCall = callContracts[1];
        var nearPut = putContracts[1];
        var farPut = putContracts.Single(x => x.Strike == nearPut.Strike - farCall.Strike + nearCall.Strike);

        // Order Strategy
        var ironCondor = OptionStrategies.IronCondor(
            _symbol,
            farPut.Strike,
            nearPut.Strike,
            nearCall.Strike,
            farCall.Strike,
            expiry);

        Buy(ironCondor, 2);
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm
IndexOptionIronCondorAlgorithm.cs C#

Option Strategies

Short Iron Condor

Introduction

The Short Iron Condor is an Option strategy that consists of four contracts. All the contracts have the same underlying Equity and expiration, but the order of strike prices is A>B>C>D. The following table describes the strike prices of each contract:

Position	Strike
-1 far-OTM call	A
1 near-OTM call	B, where B > underlying\ price
1 near-OTM put	C, where C < underlying\ price
-1 far-OTM put	D, where C-D = A-B

The short iron condor consists of selling a far OTM call, selling a far OTM put, buying a near OTM call, and buying a near OTM put. This strategy profits from a increase in price movement (implied volatility).

Implementation

Follow these steps to implement the short iron condor strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(500000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG");
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().IronCondor(30, 5, 10));
}
```

   The `IronCondor` filter narrows the universe down to just the four contracts you need to form a short iron condor.

2. In the `OnData` method, select the contracts in the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio[_symbol.Underlying].Invested)
    {
        Liquidate();
    }

    if (Portfolio.Invested || !IsMarketOpen(_symbol) ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Find put and call contracts with the farthest expiry
    var expiry = chain.Max(x => x.Expiry);
    var contracts = chain.Where(x => x.Expiry == expiry).OrderBy(x => x.Strike);

    var putContracts = contracts.Where(x => x.Right == OptionRight.Put).ToArray();
    var callContracts = contracts.Where(x => x.Right == OptionRight.Call).ToArray();

    if (putContracts.Length < 2 || callContracts.Length < 2) return;

    // Select the strategy legs
    var farPut = putContracts[0];
    var nearPut = putContracts[1];
    var nearCall = callContracts[0];
    var farCall = callContracts[1];
```

3. In the `OnData` method, place the orders.

   Approach A: Call the `OptionStrategies.ShortIronCondor` method with the details of each leg and then pass the result to the `Buy` method.

   ```C#
   var shortIronCondor = OptionStrategies.ShortIronCondor(
       _symbol,
       farPut.Strike,
       nearPut.Strike,
       nearCall.Strike,
       farCall.Strike,
       expiry);

   Buy(shortIronCondor, 2);
   ```

   Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

   ```C#
   var legs = new List<Leg>()
       {
           Leg.Create(farPut.Symbol, -1),
           Leg.Create(nearPut.Symbol, 1),
           Leg.Create(farCall.Symbol, -1),
           Leg.Create(nearCall.Symbol, 1)
       };
   ComboMarketOrder(legs, 1);
   ```

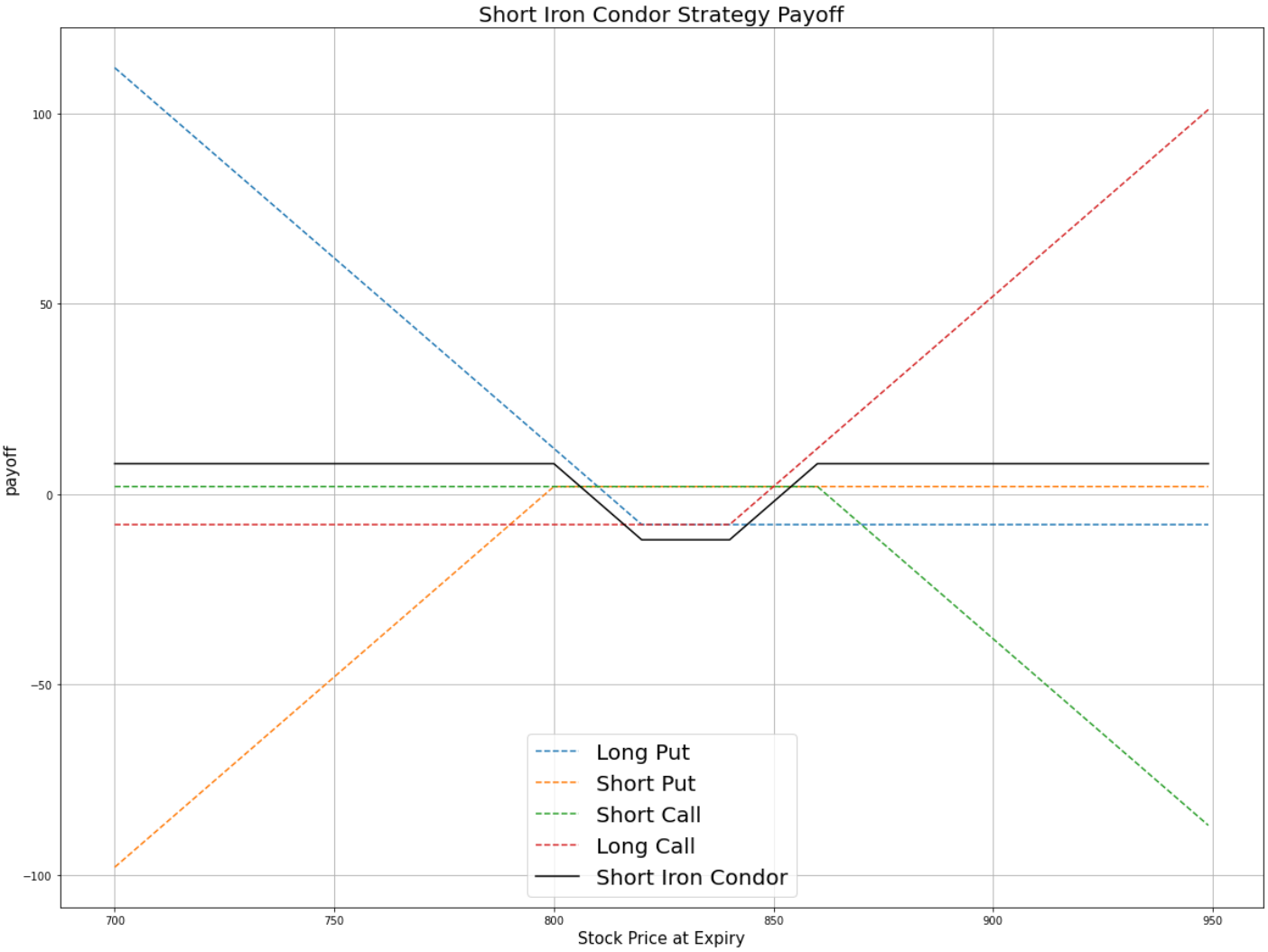
Strategy Payoff

This is a limited-reward-limited-risk strategy. The payoff is

$$\begin{array}{rcll} C^{far}_T & = & (S_T - K^C_{far})^{+}\\ C^{near}_T & = & (S_T - K^C_{near})^{+}\\ P^{far}_T & = & (K^P_{far} - S_T)^{+}\\ P^{near}_T & = & (K^P_{near} - S_T)^{+}\\ P_T & = & (C^{near}_T + P^{near}_T - C^{far}_T - P^{far}_T - C^{near}_0 - P^{near}_0 + C^{far}_0 + P^{far}_0)\times m - fee \end{array} \begin{array}{rcl} \textrm{where} & C^{far}_T & = & \textrm{Far OTM call value at time T}\\ & C^{near}_T & = & \textrm{Near OTM call value at time T}\\ & P^{far}_T & = & \textrm{Far OTM put value at time T}\\ & P^{near}_T & = & \textrm{Near ATM put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^C_{far} & = & \textrm{Far OTM call strike price}\\ & K^C_{near} & = & \textrm{Near OTM call strike price}\\ & K^P_{far} &$$

= & \textrm{Far OTM put strike price}\\ & K^P_{near} & = & \textrm{Near OTM put strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & C^{far}_0 & = & \textrm{Far OTM call value at position opening (credit received)}\\ & C^{near}_0 & = & \textrm{Near OTM call value at position opening (debit paid)}\\ & P^{far}_0 & = & \textrm{Far OTM put value at position opening (credit received)}\\ & P^{near}_0 & = & \textrm{Near OTM put value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}

The following chart shows the payoff at expiration:



The maximum profit is K^C_{far} - K^C_{near} - C^{near}_0 - P^{near}_0 + C^{far}_0 + P^{far}_0, where K^P_{OTM} > S_T or S_T > K^C_{OTM}.

The maximum loss is the net debit paid: C^{far}_0 + P^{far}_0 - C^{near}_0 - P^{near}_0, where K^P_{OTM} < S_T < K^C_{OTM}.

If the Option is American Option, there is a risk of early assignment on the contracts you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	Price ($)	Strike ($)
Far-OTM call	1.05	857.50
Far-OTM put	2.15	815.00
Near-OTM call	2.75	852.50
Near-OTM put	4.80	820.00
Underlying Equity at expiration	843.25	-

Therefore, the payoff is

\begin{array}{rcll} C^{far}_T & = & (S_T - K^C_{far})^{+}\\ & = & (843.25-857.50)^{+}\\ & = & 0\\ C^{near}_T & = & (S_T - K^C_{near})^{+}\\ & = & (843.25-852.50)^{+}\\ & = & 0\\ P^{far}_T & = & (K^P_{far} - S_T)^{+}\\ & = & (815.00-843.25)^{+}\\ & = & 0\\ P^{near}_T & = & (K^P_{near} - S_T)^{+}\\ & = & (820.00-843.25)^{+}\\ & = & 0\\ P_T & = & (C^{near}_T + P^{near}_T - C^{far}_T - P^{far}_T - C^{near}_0 - P^{near}_0 + C^{far}_0 + P^{far}_0)\times m - fee\\ & = & (0+0-0-0-2.75-4.80+1.05+2.15)\times100-1\times4\\ & = & -439 \end{array}

So, the strategy loses $439.

The following algorithm implements a short iron condor Option strategy:

C#

```
public class ShortIronCondorStrategy : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().IronCondor(30, 5, 10));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested || !IsMarketOpen(_symbol) ||
            !slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            return;
        }

        // Find put and call contracts with the farthest expiry
        var expiry = chain.Max(x => x.Expiry);
        var contracts = chain.Where(x => x.Expiry == expiry).OrderBy(x => x.Strike);

        var putContracts = contracts.Where(x => x.Right == OptionRight.Put).ToArray();
        var callContracts = contracts.Where(x => x.Right == OptionRight.Call).ToArray();

        if (putContracts.Length < 2 || callContracts.Length < 2) return;

        // Select the strategy legs
        var nearCall = callContracts[0];
        var farCall = callContracts[1];
        var nearPut = putContracts[1];
        var farPut = putContracts.Single(x => x.Strike == nearPut.Strike - farCall.Strike + nearCall.Strike);

        // Order Strategy
        var shortIronCondor = OptionStrategies.ShortIronCondor(
            _symbol,
            farPut.Strike,
            nearPut.Strike,
            nearCall.Strike,
            farCall.Strike,
            expiry);

        Buy(shortIronCondor, 2);
    }
}
```

Option Strategies

Naked Call

Introduction

A Naked Call , also known as an uncovered call , consists of selling call Options without owning the underlying asset. It's called "naked" because you don't have any cover or protection in the form of owning the underlying asset, which exposes you to potentially unlimited risk. Naked calls aim to profit from the Option premium by selling calls. At any time for American Options or at expiration for European Options, if the stock moves below the strike price, you keep the premium. If the underlying price moves above the strike, the Option buyer can exercise the Options contract , which means you need buy the underlying at market price to fulfill your obligation to sell it at the strike price and keep the premium.

Implementation

Follow these steps to implement the naked call strategy:

1. In the `Initialize` method, set the start date, end date, starting cash, and Options universe .

```
C#
private Symbol _call, _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(100000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("IBM");
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().NakedCall(30, 0));
}
```

   The `NakedCall` filter narrows the universe down to just the one contract you need to form a naked call.

2. In the `OnData` method, select the Option contract.

```
C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain)) return;

    // Find ATM call with the farthest expiry
    var expiry = chain.Max(x => x.Expiry);
    var atmCall = chain
        .Where(x => x.Right == OptionRight.Call && x.Expiry == expiry)
        .OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price))
        .FirstOrDefault();
```

3. In the `OnData` method, place the orders.

   Approach A: Call the `OptionStrategies.NakedCall` method with the details of each leg and then pass the result to the `Buy` method.

```C#
var nakedCall = OptionStrategies.NakedCall(_symbol, atmCall.Strike, expiry);
Buy(nakedCall, 1);
```

Approach B: Call the Market Order or Limit Order method.

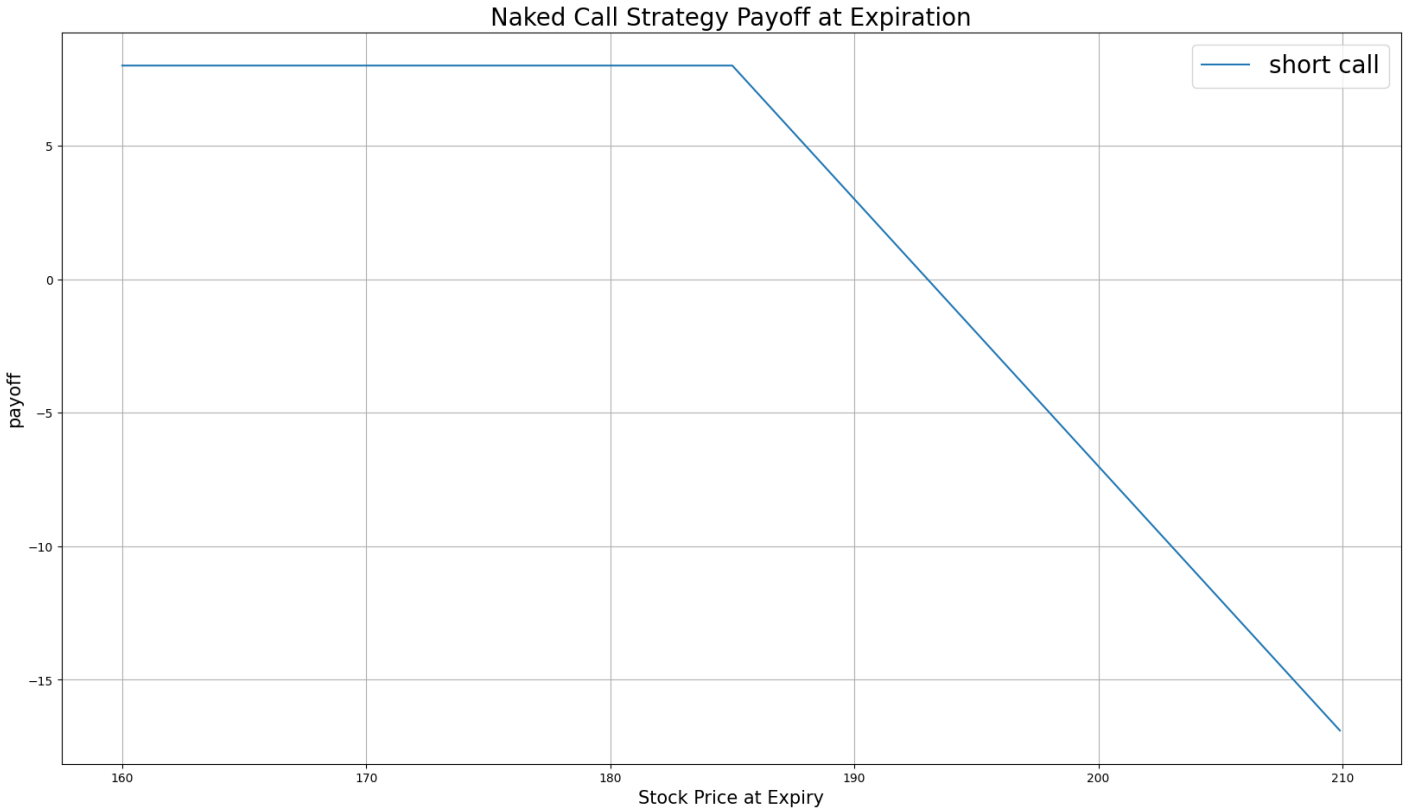
```C#
MarketOrder(atmCall.Symbol, -1);
```

Strategy Payoff

The payoff of the strategy is

$$\begin{array}{rcl} C^{K}_T & = & (S_T - K)^{+}\\ P_T & = & (C^{K}_0 - C^{K}_T)\times m - fee \end{array} \begin{array}{rcl} \textrm{where} & C^{K}_T & = & \textrm{Call value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K & = & \textrm{Call strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & C^{K}_0 & = & \textrm{Call price when the trade opened (credit received)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is C^{K}_0, which occurs when the underlying price is at or below the strike price of the call at expiration.

The maximum loss is unlimited because there is no limit to how high the underlying asset's price can rise.

If the Option is American Option, there is a risk of early assignment on the contract you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	Price ($)	Strike ($)
Call	3.35	185.00
Underlying Equity at expiration	190.01	-

Therefore, the payoff is

$$\begin{array}{rcll} C^{K}_T & = & (S_T - K)^{+}\\ & = & (190.01 - 185)^{+}\\ & = & 5.01\\ P_T & = & (C^{K}_0 - C^{K}_T)\times m - fee\\ & = & (3.35 - 5.01)\times m - fee\\ & = & -1.66 \times 100 - 2\\ & = & -167 \end{array}$$

So, the strategy loses $167.

The following algorithm implements a naked call Option strategy:

```C#
public class NakedCallAlgorithm : QCAlgorithm
{
    private Symbol _call, _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("IBM");
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().NakedCall(30, 0));

        // use the underlying equity as the benchmark
        SetBenchmark(_symbol.Underlying);
    }

    public override void OnData(Slice slice)
    {
        if (_call != null && Portfolio[_call].Invested) return;

        if (!slice.OptionChains.TryGetValue(_symbol, out var chain)) return;

        // Find ATM call with the farthest expiry
        var expiry = chain.Max(x => x.Expiry);
        var atmCall = chain
            .Where(x=> x.Right == OptionRight.Call && x.Expiry == expiry)
            .OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price))
            .FirstOrDefault();

        if (atmCall == null) return;

        var nakedCall = OptionStrategies.NakedCall(_symbol, atmCall.Strike, expiry);
        Buy(nakedCall, 1);

        _call = atmCall.Symbol;
    }
}
```

Option Strategies

Naked Put

Introduction

A Naked Put , also known as an uncovered put , consists of selling put Options without having a corresponding short position in the underlying security. It's called "naked" because you don't have any cover or protection in the form of a short position in the underlying asset, which exposes you to potentially unlimited risk. Naked puts aim to profit from the Option premium by selling puts. At any time for American Options or at expiration for European Options, if the stock moves above the strike price, you keep the premium. If the underlying price moves below the strike, the Option buyer can exercise the Options contract , which means you need sell the underlying at market price to fulfill your obligation to buy it at the strike price and keep the premium.

Implementation

Follow these steps to implement the naked put strategy:

1. In the `Initialize` method, set the start date, end date, starting cash, and Options universe .

```
C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(100000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("IBM");
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().NakedPut(30, 0));
}
```

   The `NakedPut` filter narrows the universe down to just the one contract you need to form a naked put.

2. In the `OnData` method, select the Option contract.

```
C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain)) return;

    // Find ATM put with the farthest expiry
    var expiry = chain.Max(x => x.Expiry);
    var atmPut = chain
        .Where(x => x.Right == OptionRight.Put && x.Expiry == expiry)
        .OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price))
        .FirstOrDefault();
```

3. In the `OnData` method, place the orders.

   Approach A: Call the `OptionStrategies.NakedPut` method with the details of each leg and then pass the result to the `Buy` method.

```C#
var nakedPut = OptionStrategies.NakedPut(_symbol, atmPut.Strike, expiry);
Buy(nakedPut, 1);
```

Approach B: Call the Market Order or Limit Order method.

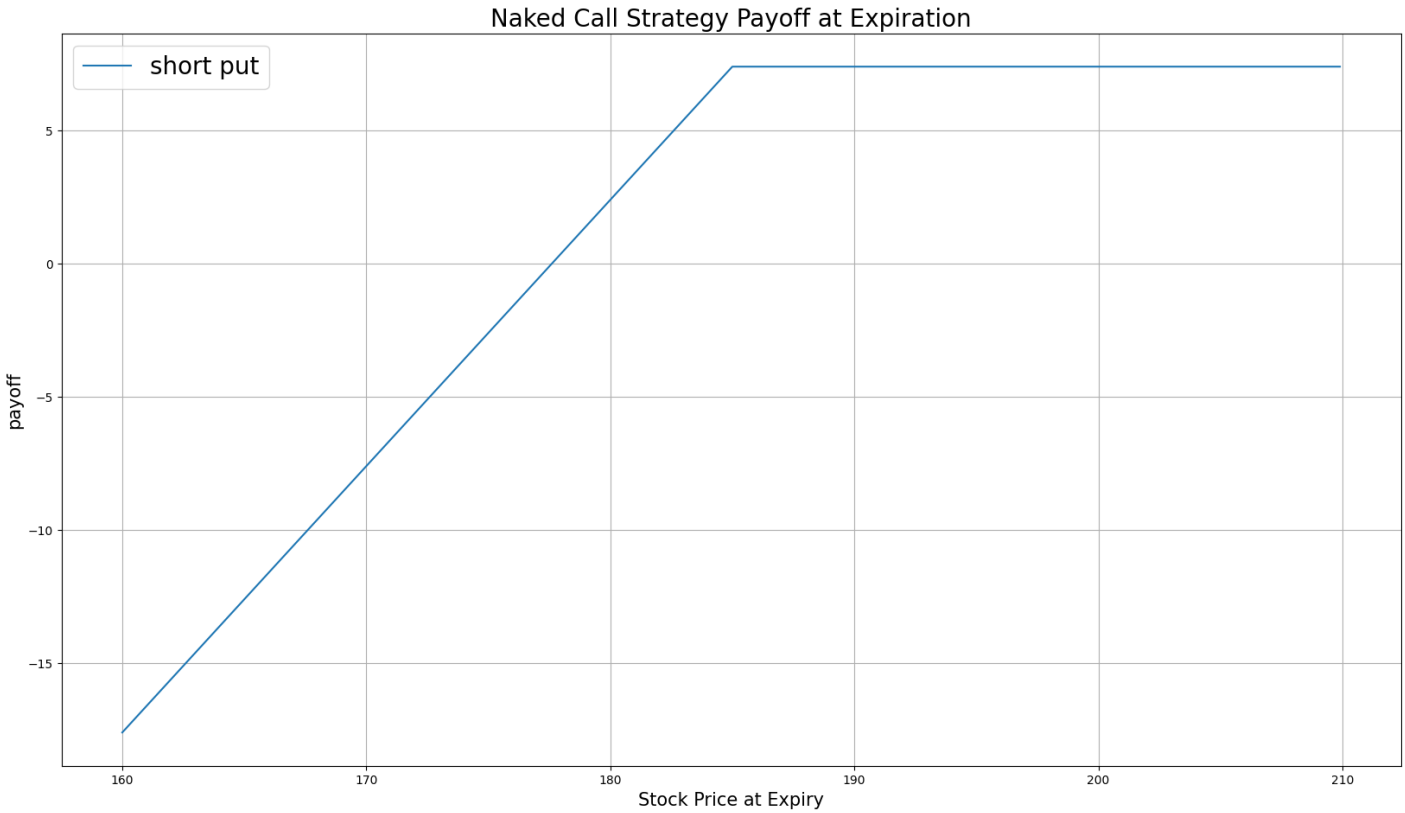
```C#
MarketOrder(atmPut.Symbol, -1);
```

Strategy Payoff

The payoff of the strategy is

$$\begin{array}{rcl} P^{K}_T & = & (K - S_T)^{+}\\ P_T & = & (P^{K}_0 - P^{K}_T)\times m - fee \end{array} \begin{array}{rcl} \textrm{where} & P^{K}_T & = & \textrm{Put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K & = & \textrm{Put strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & P^{K}_0 & = & \textrm{Put price when the trade opened (credit received)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is P^{K}_0, which occurs when the underlying price is at or above the strike price of the put at expiration.

The maximum loss is $K - P^{K}_0$, which ocurrs when the underlying price drops to zero.

If the Option is American Option, there is a risk of early assignment on the contract you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	Price ($)	Strike ($)
Put	1.37	185.00
Underlying Equity at expiration	190.01	-

Therefore, the payoff is

$$\begin{array}{rcll} P^{K}_T & = & (K - S_T)^{+}\\ & = & (185 - 190.01)^{+}\\ & = & 0.0\\ P_T & = & (P^{K}_0 - P^{K}_T)\times m - fee\\ & = & (1.37 - 0.0)\times m - fee\\ & = & 1.37 \times 100 - 1\\ & = & 136 \end{array}$$

So, the strategy gains $136.

The following algorithm implements a naked put Option strategy:

```C#
public class NakedPutAlgorithm : QCAlgorithm
{
    private Symbol _put, _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("IBM");
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().NakedPut(30, 0));

        // use the underlying equity as the benchmark
        SetBenchmark(_symbol.Underlying);
    }

    public override void OnData(Slice slice)
    {
        if (_put != null && Portfolio[_put].Invested) return;

        if (!slice.OptionChains.TryGetValue(_symbol, out var chain)) return;

        // Find ATM put with the farthest expiry
        var expiry = chain.Max(x => x.Expiry);
        var atmPut = chain
            .Where(x=> x.Right == OptionRight.Put && x.Expiry == expiry)
            .OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price))
            .FirstOrDefault();

        if (atmPut == null) return;

        var nakedPut = OptionStrategies.NakedPut(_symbol, atmPut.Strike, expiry);
        Buy(nakedPut, 1);

        _put = atmPut.Symbol;
    }
}
```

Option Strategies

Protective Call

Introduction

A Protective Call consists of a short position in a stock and a long position in a call Option for the same amount of stock. Protective calls aim to hedge the short position of a stock with a long ATM or slightly OTM call Option. At any time for American Options or at expiration for European Options, if the stock moves below the strike price, the Option contract becomes worthless but the short position acquires an unrealized gain. If the underlying price moves above the strike, you can exercise the Options contract and receive the underlying Equity, which closes your short position.

Implementation

Follow these steps to implement the protective call strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Options universe .

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(100000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("IBM");
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().NakedCall(30, 0));
}
```

   The `NakedCall` filter narrows the universe down to just the one contract you need to form a protective call.

2. In the `OnData` method, select the Option contract.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain)) return;

    // Find ATM call with the farthest expiry
    var expiry = chain.Max(x => x.Expiry);
    var atmCall = chain
        .Where(x => x.Right == OptionRight.Call && x.Expiry == expiry)
        .OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price))
        .FirstOrDefault();
```

3. In the `OnData` method, place the orders.

   Approach A: Call the `OptionStrategies.ProtectiveCall` method with the details of each leg and then pass the result to the `Buy` method.

```C#
var protectiveCall = OptionStrategies.ProtectiveCall(_symbol, atmCall.Strike, expiry);
Buy(protectiveCall, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

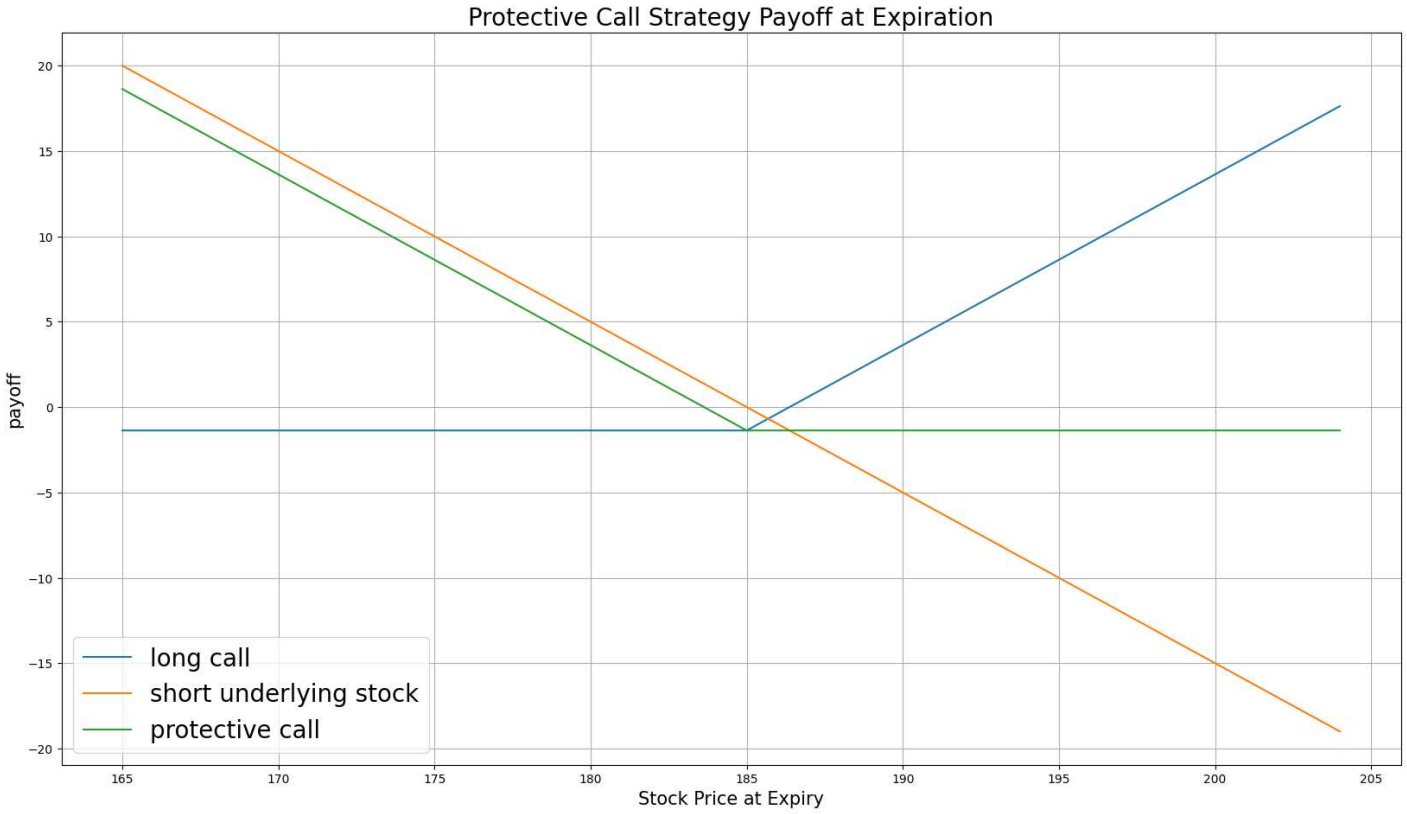
```C#
var legs = new List<Leg>()
    {
        Leg.Create(atmCall.Symbol, 1),
        Leg.Create(chain.Underlying.Symbol, -chain.Underlying.SymbolProperties.ContractMultiplier)
    };
ComboMarketOrder(legs, 1);
```

Strategy Payoff

The payoff of the strategy is

$$\begin{array}{rcll} C^{K}_T & = & (S_T - K)^{+}\\ P_T & = & (S_0 - S_T + C^{K}_T - C^{K}_0)\times m - fee \end{array}$$
$$\begin{array}{rcll} \textrm{where} & C^{K}_T & = & \textrm{Call value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K & = & \textrm{Call strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & S_0 & = & \textrm{Underlying asset price when the trade opened}\\ & C^{K}_0 & = & \textrm{Call price when the trade opened (credit received)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is $S_0 - C^{K}_0$, which occurs when the underlying price is 0.

The maximum loss is $S_0 - K - C^{K}_0$, which occurs when the underlying price is above the strike price.

Example

The following table shows the price details of the assets in the algorithm:

Asset	Price ($)	Strike ($)
Call	3.50	185.00
Underlying Equity at start of the trade	186.94	-
Underlying Equity at expiration	190.01	-

Therefore, the payoff is

$$\begin{array}{rcll} C^{K}_T & = & (S_T - K)^{+}\\ & = & (190.01 - 185)^{+}\\ & = & 5.01\\ P_T & = & (S_0 - S_T + C^{K}_T - C^{K}_0)\times m - fee\\ & = & (186.94 - 190.01 + 5.01 - 3.50)\times m - fee\\ & = & -1.56 \times 100 - 2\\ & = & -158 \end{array}$$

So, the strategy loses $158.

The following algorithm implements a protective call Option strategy:

```C#
public class ProtectiveCallAlgorithm : QCAlgorithm
{
    private Symbol _call, _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("IBM");
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().NakedCall(30, 0));

        // use the underlying equity as the benchmark
        SetBenchmark(_symbol.Underlying);
    }

    public override void OnData(Slice slice)
    {
        if (_call != null && Portfolio[_call].Invested) return;

        if (!slice.OptionChains.TryGetValue(_symbol, out var chain)) return;

        // Find ATM call with the farthest expiry
        var expiry = chain.Max(x => x.Expiry);
        var atmCall = chain
            .Where(x=> x.Right == OptionRight.Call && x.Expiry == expiry)
            .OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price))
            .FirstOrDefault();

        if (atmCall == null) return;

        var protectiveCall = OptionStrategies.ProtectiveCall(_symbol, atmCall.Strike, expiry);
        Buy(protectiveCall, 1);

        _call = atmCall.Symbol;
    }
}
```

Option Strategies

Protective Put

Introduction

A Protective Put consists of buying a long position in a stock and a long position in put Options for the same amount of stock. Protective puts aim to hedge the long position of a stock with a long ATM or slightly OTM put Option. At any time for American Options or at expiration for European Options, if the stock moves above the strike price, the Option contract becomes worthless but the long stock position acquires an unrealized gain. If the underlying price moves below the strike, you can exercise the Options contract , which sells your underlying position at the put Option strike price.

Implementation

Follow these steps to implement the protective put strategy:

1. In the `Initialize` method, set the start date, end date, starting cash, and Options universe .

```
C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(100000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("IBM");
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().NakedPut(30, 0));
}
```

   The `NakedPut` filter narrows the universe down to just the one contract you need to form a protective put.

2. In the `OnData` method, select the Option contract.

```
C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Find ATM put with the farthest expiry
    var expiry = chain.Max(x => x.Expiry);
    var atmPut = chain
        .Where(x => x.Right == OptionRight.Put && x.Expiry == expiry)
        .OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price))
        .FirstOrDefault();
```

3. In the `OnData` method, place the orders.

   Approach A: Call the `OptionStrategies.ProtectivePut` method with the details of each leg and then pass the result to the `Buy` method.

```
C#
var protectivePut = OptionStrategies.ProtectivePut(_symbol, atmPut.Strike, expiry);
Buy(protectivePut, 1);
```

   Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

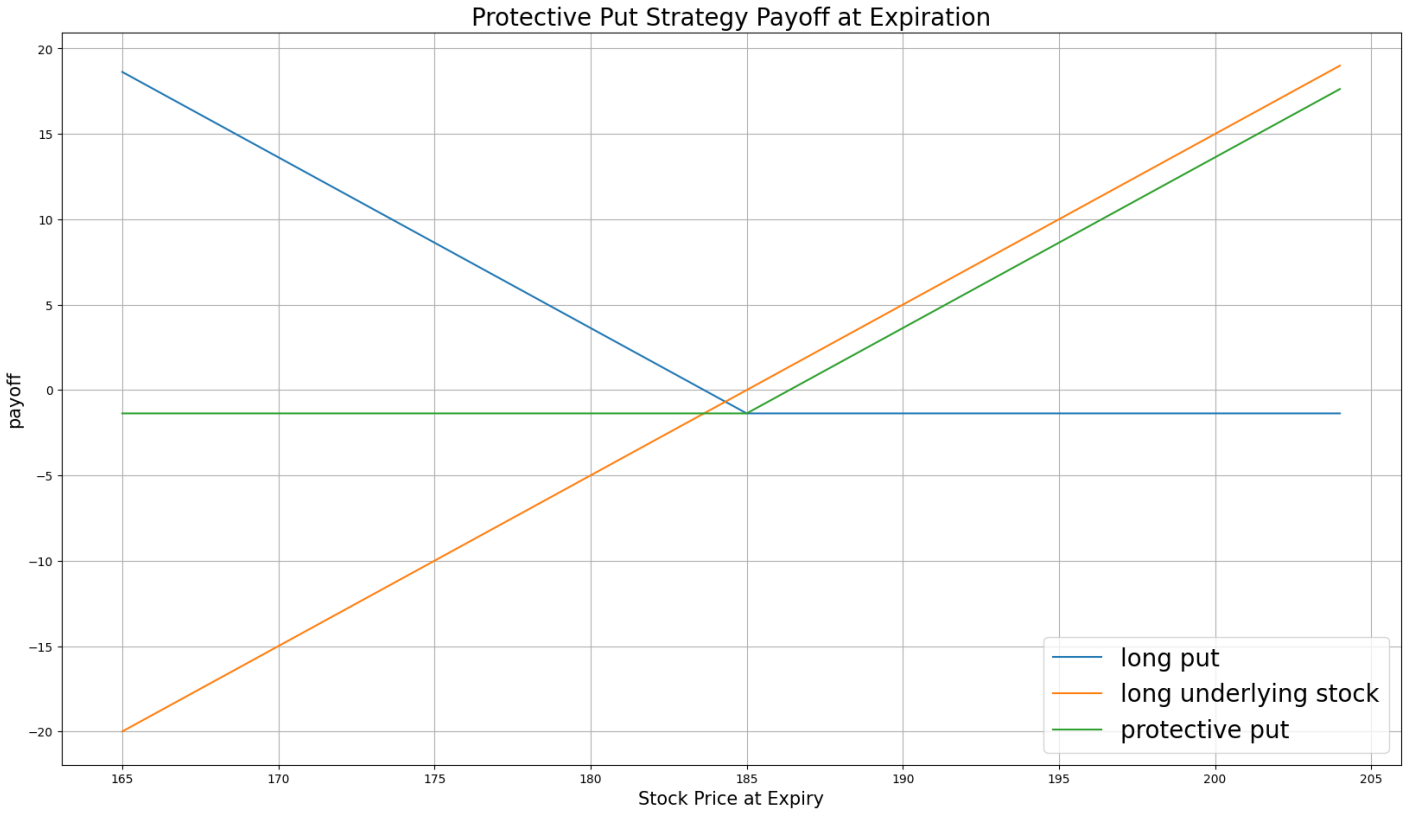
```
C#
var legs = new List<Leg>()
    {
        Leg.Create(atmPut.Symbol, 1),
        Leg.Create(chain.Underlying.Symbol, chain.Underlying.SymbolProperties.ContractMultiplier)
    };
ComboMarketOrder(legs, 1);
```

Strategy Payoff

The payoff of the strategy is

$$\begin{array}{rcll} P^{K}_T & = & (K - S_T)^{+}\\ P_T & = & (S_T - S_0 + P^{K}_T - P^{K}_0)\times m - fee \end{array}$$
$$\begin{array}{rcll} \textrm{where} & P^{K}_T & = & \textrm{Put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K & = & \textrm{Put strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & S_0 & = & \textrm{Underlying asset price when the trade opened}\\ & P^{K}_0 & = & \textrm{Put price when the trade opened (credit received)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is $S_T - S_0 - P^{K}_0$, which occurs when the underlying price is above the $S_0 + P^{K}_0$.

The maximum loss is P^{K}_0, which occurs when the underlying price drops.

Example

The following table shows the price details of the assets in the algorithm:

Asset	Price ($)	Strike ($)
Put	1.53	185.00
Underlying Equity at start of the trade	187.07	-
Underlying Equity at expiration	190.01	-

Therefore, the payoff is

$$\begin{array}{rcll} P^{K}_T & = & (K - S_T)^{+}\\ & = & (185 - 190.1)^{+}\\ & = & 0\\ P_T & = & (S_T - S_0 + P^{K}_T - P^{K}_0)\times m - fee\\ & = & (190.01 - 187.07 + 0 - 1.53)\times m - fee\\ & = & 1.41 \times 100 - 2\\ & = & 139 \end{array}$$

So, the strategy gains $139.

The following algorithm implements a protective put Option strategy:

C#

```
public class ProtectivePutAlgorithm : QCAlgorithm
{
    private Symbol _put, _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("IBM");
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().NakedPut(30, 0));

        // use the underlying equity as the benchmark
        SetBenchmark(_symbol.Underlying);
    }

    public override void OnData(Slice slice)
    {
        if (_put != null && Portfolio[_put].Invested) return;

        if (!slice.OptionChains.TryGetValue(_symbol, out var chain)) return;

        // Find ATM put with the farthest expiry
        var expiry = chain.Max(x => x.Expiry);
        var atmPut = chain
            .Where(x=> x.Right == OptionRight.Put && x.Expiry == expiry)
            .OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price))
            .FirstOrDefault();

        if (atmPut == null) return;

        var protectivePut = OptionStrategies.ProtectivePut(_symbol, atmPut.Strike, expiry);
        Buy(protectivePut, 1);

        _put = atmPut.Symbol;
    }
}
```

Option Strategies

Protective Collar

Introduction

A Protective Collar is an Options strategy that consists of a covered call and a long put (protective put) with a lower strike price than the short call contract. In contrast to the covered call, the protective put component limits the drawdown of the strategy when the underlying price decreases too much.

Implementation

Follow these steps to implement the protective collar strategy:

1. In the `Initialize` method, set the start date, set the end date, subscribe to the underlying Equity , and create an Option universe .

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(100000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().ProtectiveCollar(30, -1, -10));
}
```

   The `ProtectiveCollar` filter narrows the universe down to just the two contracts you need to form a protective collar.

2. In the `OnData` method, select the Option contracts.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested) return;

    // Get the OptionChain
    if (!slice.OptionChains.TryGetValue(_symbol, out var chain)) return;

    // Select an expiry date
    var expiry = chain.Max(x => x.Expiry);

    // Select the call and put contracts that expire on the selected date
    var calls = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Call);
    var puts = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Put);
    if (calls.Count() == 0 || puts.Count() == 0) return;

    // Select the OTM contracts
    var call = calls.OrderBy(x => x.Strike).Last();
    var put = puts.OrderBy(x => x.Strike).First();
```

3. In the `OnData` method, select the contracts and place the orders.

   Approach A: Call the `OptionStrategies.ProtectiveCollar` method with the details of each leg and then pass the result to the `Buy` method.

```C#
var protectiveCollar = OptionStrategies.ProtectiveCollar(_symbol, call.Strike, put.Strike, expiry);
Buy(protectiveCollar, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

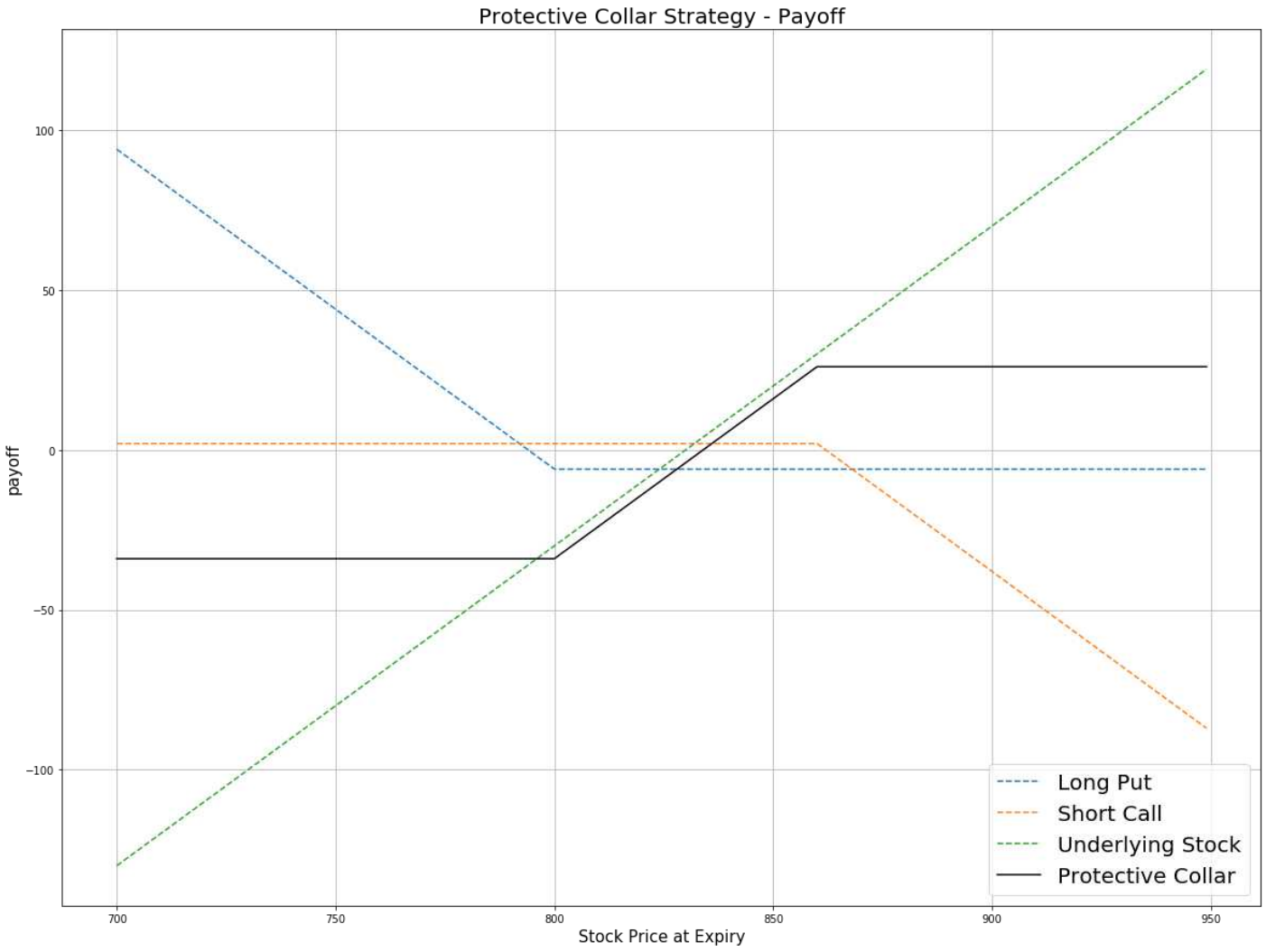
```C#
var legs = new List<Leg>()
    {
        Leg.Create(call.Symbol, -1),
        Leg.Create(put.Symbol, 1),
        Leg.Create(chain.Underlying.Symbol, chain.Underlying.SymbolProperties.ContractMultiplier)
    };
ComboMarketOrder(legs, 1);
```

Strategy Payoff

This is a limited-profit-limited-loss strategy. The payoff is

$$\begin{array}{rcll} C_T & = & (S_T - K^{C})^{+}\\ P_T & = & (K^{P} - S_T)^{+}\\ Payoff_T & = & (S_T - S_0 - C_T + P_T + C_0 - P_0)\times m - fee \end{array} \begin{array}{rcl} \textrm{where} & C_T & = & \textrm{Call value at time T}\\ & P_T & = & \textrm{Put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^{C} & = & \textrm{Call strike price}\\ & K^{P} & = & \textrm{Put strike price}\\ & Payoff_T & = & \textrm{Payout total at time T}\\ & S_0 & = & \textrm{Underlying asset price when the trade opened}\\ & C_0 & = & \textrm{Call price when the trade opened (credit received)}\\ & P_0 & = & \textrm{Put price when the trade opened (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is $K^{C} - S_T + C_0 - P_0$. It occurs when the underlying price is at or above the strike price of the call at expiration.

The maximum profit is $S_T - K^{P} + C_0 - P_0$. It occurs when the underlying price is at or below the strike price of the put at expiration.

If the Option is American Option, there is a risk of early assignment on the contract you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	Price ($)	Strike ($)
Call	2.85	845.00
Put	6.00	822.50
Underlying Equity at position opens	833.17	-
Underlying Equity at expiration	843.25	-

Therefore, the payoff is

$$\begin{array}{rcll} C_T & = & (S_T - K^{C})^{+}\\ & = & (843.365 - 845.00)^{+}\\ & = & 0\\ P_T & = & (K^{P} - S_T)^{+}\\ & = & (822.50 - 843.365)^{+}\\ & = & 0\\ Payoff_T & = & (S_T - S_0 - C_T + P_T + C_0 - P_0)\times m - fee\\ & = & (843.25 - 833.17 - 0 + 0 + 2.85 - 6.00)\times100-1.00\times3\\ & = & 690\\ \end{array}$$

So, the strategy gains $690.

The following algorithm implements a protective collar Option strategy:

C#

```
public class ProtectiveCollarStrategy : QCAlgorithm
{
    private Symbol _equity;
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        _equity = AddEquity("GOOG", Resolution.Minute).Symbol;
        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().ProtectiveCollar(30, -1, -10));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested) return;

        // Get the OptionChain of the symbol
        var chain = slice.OptionChains.get(_symbol, null);
        if (chain == null || chain.Count() == 0) return;

        // sort the optionchain by expiration date and choose the furthest date
        var expiry = chain.OrderBy(x => x.Expiry).Last().Expiry;

        // filter the call and put options from the contracts expires on that date
        var calls = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Call);
        var puts = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Put);
        if (calls.Count() == 0 || puts.Count() == 0) return;

        // select the strike prices
        var callStrike = calls.OrderBy(x => x.Strike).Last().Strike;
        var putStrike = puts.OrderBy(x => x.Strike).First().Strike;

        var protectiveCollar = OptionStrategies.ProtectiveCollar(_symbol, callStrike, putStrike, expiry);
        Buy(protectiveCollar, 1);
    }

    public override void OnEndOfDay(Symbol symbol)
    {
        if (symbol.Value == "GOOG")
        {
            Log($"{Time}::{symbol}::{Securities[symbol].Price}");
        }
    }
}
```

Option Strategies

Long Straddle

Introduction

Long Straddle is an Options trading strategy that consists of buying an ATM call and an ATM put, where both contracts have the same underlying asset, strike price, and expiration date. This strategy aims to profit from volatile movements in the underlying stock, either positive or negative.

Implementation

Follow these steps to implement the long straddle strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(100000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG");
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().Straddle(30));
}
```

   The `Straddle` filter narrows the universe down to just the two contracts you need to form a long straddle.

2. In the `OnData` method, select the expiration date and strike price of the contracts in the strategy legs.

```
C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Find ATM options with the nearest expiry
    var expiry = chain.Min(contract => contract.Expiry);
    var strike = chain.OrderBy(contract => Math.Abs(contract.Strike - chain.Underlying.Price)).First().Strike;
```

3. In the `OnData` method, select the contracts and place the orders.

   Approach A: Call the `OptionStrategies.Straddle` method with the details of each leg and then pass the result to the `Buy` method.

```
C#
var longStraddle = OptionStrategies.Straddle(_symbol, strike, expiry);
Buy(longStraddle, 1);
```

   Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit

Order method.

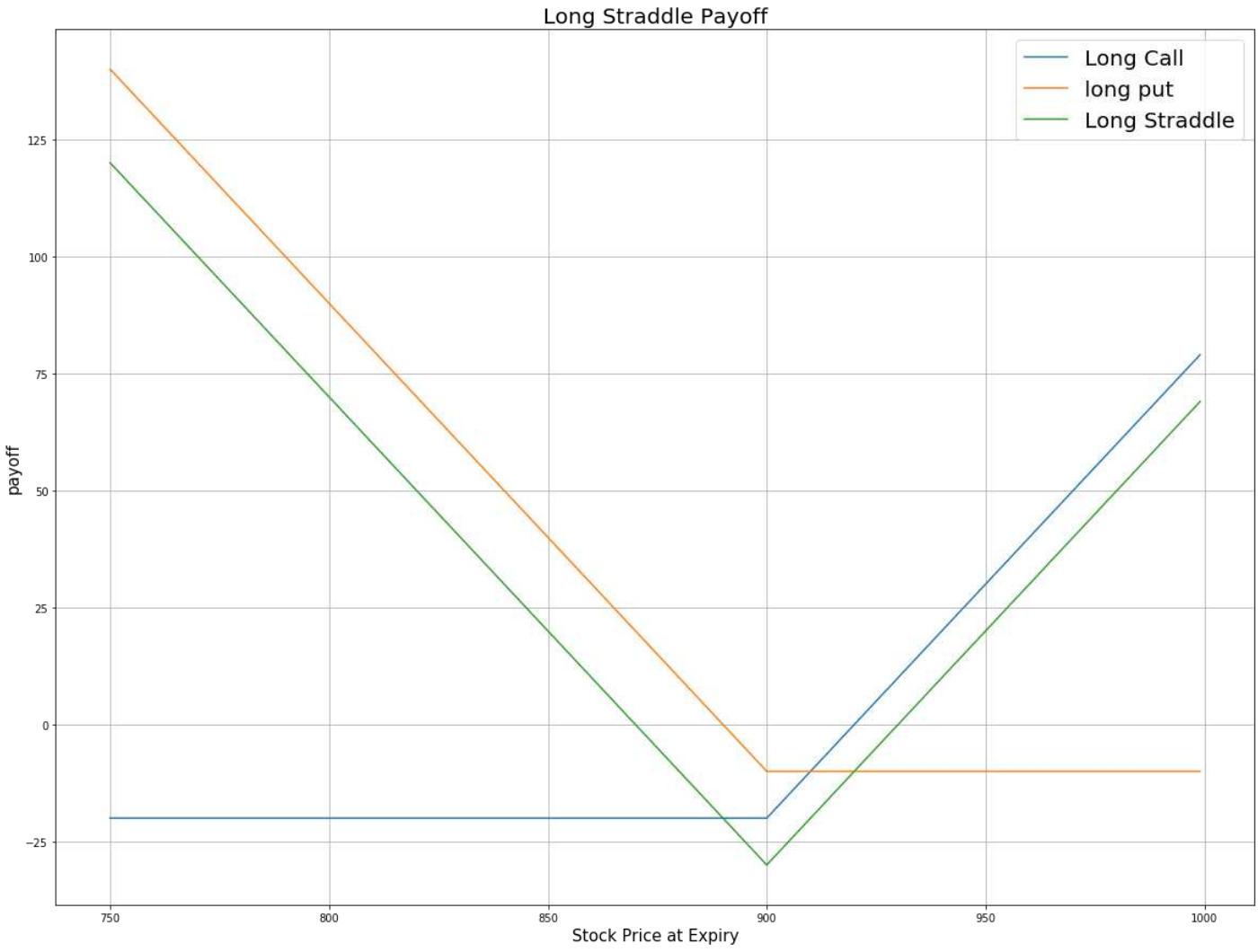
```C#
var contracts = chain.Where(contract => contract.Expiry == expiry && contract.Strike == strike);
if (contracts.Length < 2) return;

var atmCall = contracts.Single(x => x.Right == OptionRight.Call);
var atmPut = contracts.Single(x => x.Right == OptionRight.Put);

var legs = new List<Leg>()
    {
        Leg.Create(atmCall.Symbol, 1),
        Leg.Create(atmPut.Symbol, 1)
    };
ComboMarketOrder(legs, 1);
```

Strategy Payoff

The payoff of the strategy is

$$\begin{array}{rcll} C^{ATM}_T & = & (S_T - K^{C})^{+}\\ P^{ATM}_T & = & (K^{P} - S_T)^{+}\\ P_T & = & (C^{ATM}_T + P^{ATM}_T - C^{ATM}_0 - P^{ATM}_0)\times m - fee \end{array} \begin{array}{rcl} \textrm{where} & C^{ATM}_T & = & \textrm{ATM call value at time T}\\ & P^{ATM}_T & = & \textrm{ATM put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^{C} & = & \textrm{ATM call strike price}\\ & K^{P} & = & \textrm{ATM put strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & C^{ATM}_0 & = & \textrm{ATM call value at position opening (debit paid)}\\ & P^{ATM}_0 & = & \textrm{ATM put value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is unlimited if the underlying price rises to infinity or substantial, $K^{P} - C^{OTM}_0 - P^{OTM}_0$, if it drops to zero at expiration.

The maximum loss is the net debit paid, $C^{ATM}_0 + P^{ATM}_0$. It occurs when the underlying price is the same at expiration as it was when you opened the trade. In this case, both Options expire worthless.

Example

The following table shows the price details of the assets in the algorithm at Option expiration (2017-05-20):

Asset	**Price ($)**	**Strike ($)**
Call	22.30	835.00
Put	23.90	835.00
Underlying Equity at expiration	934.01	-

Therefore, the payoff is

$$\begin{array}{rcll} C^{ATM}_T & = & (S_T - K^{C})^{+}\\ & = & (934.01-835.00)^{+}\\ & = & 98.99\\ P^{ATM}_T & = & (K^{P} - S_T)^{+}\\ & = & (835.00-934.01)^{+}\\ & = & 0\\ P_T & = & (C^{ATM}_T + P^{ATM}_T - C^{ATM}_0 - P^{ATM}_0)\times m - fee\\ & = & (98.99+0-22.3-23.9)\times100-1.00\times2\\ & = & 5277 \end{array}$$

So, the strategy gains $5,277.

The following algorithm implements a long straddle Option strategy:

```C#
public class LongStraddleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("GOOG");
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().Straddle(30));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested ||
            !slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            return;
        }

        // Find ATM options with the nearest expiry
        var expiry = chain.Min(contract => contract.Expiry);
        var contracts = chain.Where(contract => contract.Expiry == expiry)
                             .OrderBy(contract => Math.Abs(contract.Strike - chain.Underlying.Price))
                             .ToArray();

        if (contracts.Length < 2) return;

        var longStraddle = OptionStrategies.Straddle(_symbol, contracts[0].Strike, expiry);
        Buy(longStraddle, 1);
    }
}
```

Option Strategies

Short Straddle

Introduction

A Short Straddle consists of selling a call and a put, where both contracts have the same underlying asset, strike price (normally at-the-money), and expiration date. If you enter a short straddle, you bet that the underlying asset will remain relatively stable and not experience significant price movements before the Option expires.

Implementation

Follow these steps to implement the short straddle strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(100000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG");
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().Straddle(30));
}
```

   The `Straddle` filter narrows the universe down to just the two contracts you need to form a short straddle.

2. In the `OnData` method, select the expiration date and strike price of the contracts in the strategy legs.

```
C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Find ATM options with the nearest expiry
    var expiry = chain.Min(contract => contract.Expiry);
    var strike = chain.OrderBy(contract => Math.Abs(contract.Strike - chain.Underlying.Price)).First().Strike;
```

3. In the `OnData` method, select the contracts and place the orders.

   Approach A: Call the `OptionStrategies.ShortStraddle` method with the details of each leg and then pass the result to the `Buy` method.

```
C#
var shortStraddle = OptionStrategies.ShortStraddle(_symbol, contracts[0].Strike, expiry);
Buy(shortStraddle, 1);
```

   Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit

Order method.

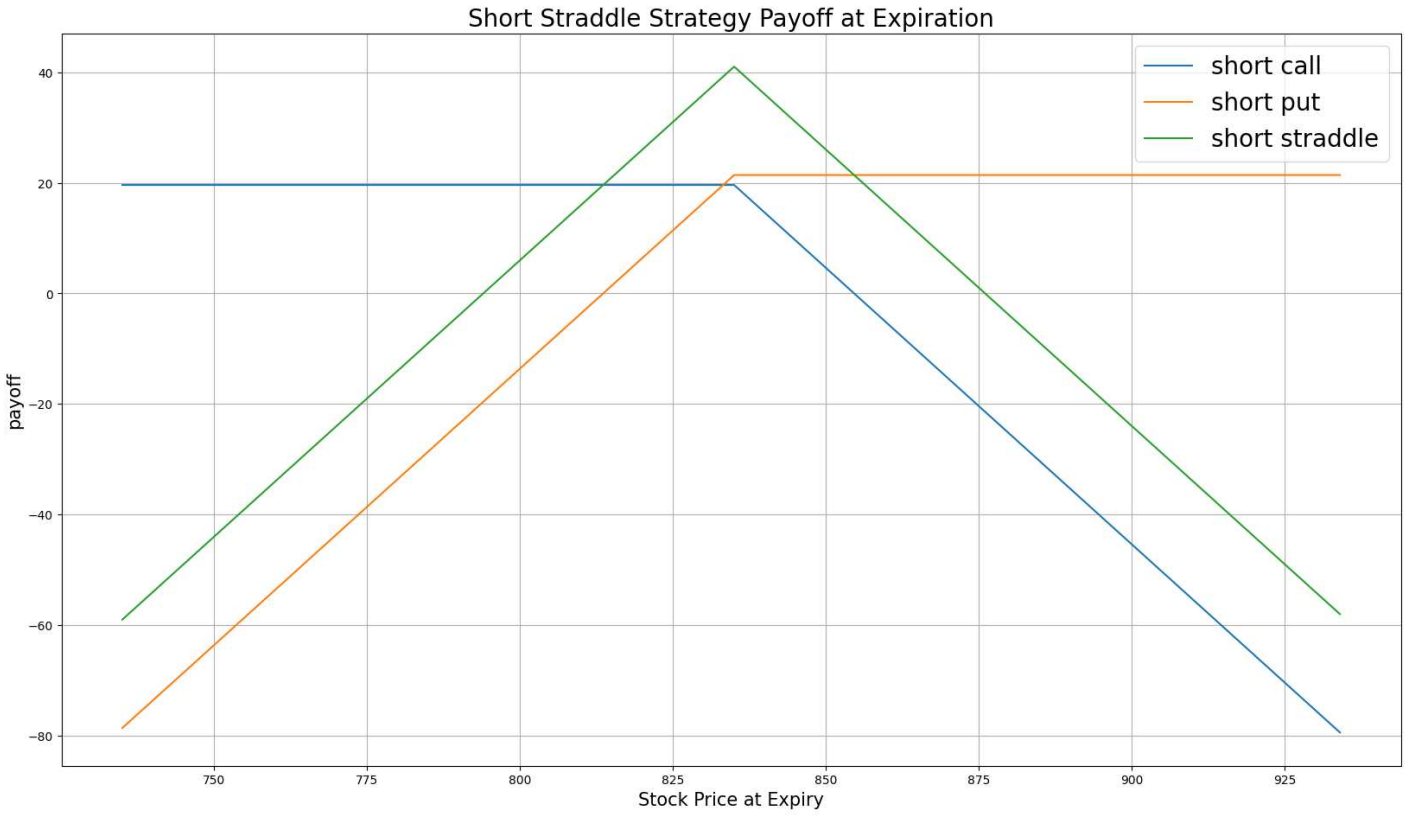
```C#
var contracts = chain.Where(contract => contract.Expiry == expiry && contract.Strike == strike);
if (contracts.Length < 2) return;

var atmCall = contracts.Single(x => x.Right == OptionRight.Call);
var atmPut = contracts.Single(x => x.Right == OptionRight.Put);

var legs = new List<Leg>()
    {
        Leg.Create(atmCall.Symbol, -1),
        Leg.Create(atmPut.Symbol, -1)
    };
ComboMarketOrder(legs, 1);
```

Strategy Payoff

The payoff of the strategy is

$$\begin{array}{rcl} C^{ATM}_T & = & (S_T - K^{C})^{+}\\ P^{ATM}_T & = & (K^{P} - S_T)^{+}\\ P_T & = & (-C^{ATM}_T - P^{ATM}_T + C^{ATM}_0 + P^{ATM}_0)\times m - fee \end{array} \begin{array}{rcl} \textrm{where} & C^{ATM}_T & = & \textrm{ATM call value at time T}\\ & P^{ATM}_T & = & \textrm{ATM put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^{C} & = & \textrm{ATM call strike price}\\ & K^{P} & = & \textrm{ATM put strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & C^{ATM}_0 & = & \textrm{ATM call value at position opening (debit paid)}\\ & P^{ATM}_0 & = & \textrm{ATM put value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is $C^{ATM}_0 + P^{ATM}_0$. It occurs when the price of the underlying asset remains exactly at the strike price upon expiration.

The maximum loss is unlimited if the underlying price rises to infinity or drops to zero at expiration.

If the Option is American Option, there is a risk of early assignment on the contracts you sell.

Example

The following table shows the price details of the assets in the algorithm at Option expiration (2017-05-20):

Asset	Price ($)	Strike ($)
Call	19.6	835.00
Put	21.4	835.00
Underlying Equity at early (2017-05-15) call assignment	932.22	-
Underlying Equity at expiration	934.01	-

Therefore, the payoff is

$$\begin{array}{rcll} C^{ATM}_T & = & (S_T - K^{C})^{+}\\ & = & (934.01-835.00)^{+}\\ & = & 99.01\\ P^{ATM}_T & = & (K^{P} - S_T)^{+}\\ & = & (835.00-934.01)^{+}\\ & = & 0\\ P_T & = & (-C^{ATM}_T - P^{ATM}_T + C^{ATM}_0 + P^{ATM}_0)\times m - fee\\ & = & (-99.01-0+19.6+21.4)\times 100 - 2.00\\ & = & -5803 \end{array}$$

So, the strategy loses $5,277. The early assigment doesn't influence the payoff.

The following algorithm implements a short straddle Option strategy:

C#

```
public class ShortStraddleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("GOOG");
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().Straddle(30));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested ||
            !slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            return;
        }

        // Find ATM options with the nearest expiry
        var expiry = chain.Min(contract => contract.Expiry);
        var contracts = chain.Where(contract => contract.Expiry == expiry)
                             .OrderBy(contract => Math.Abs(contract.Strike - chain.Underlying.Price))
                             .ToArray();

        if (contracts.Length < 2) return;

        var shortStraddle = OptionStrategies.ShortStraddle(_symbol, contracts[0].Strike, expiry);
        Buy(shortStraddle, 1);
    }
}
```

Option Strategies

Long Strangle

Introduction

Long Strangle is an Options trading strategy that consists of simultaneously buying an OTM put and an OTM call, where both contracts have the same underlying asset and expiration date. This strategy aims to profit from volatile movements in the underlying stock, either positive or negative.

Compared to a long straddle , the net debit of a long strangle is lower since OTM Options are cheaper. Additionally, the losing range of a long straddle is wider and the strike spread is wider.

Implementation

Follow these steps to implement the long strangle strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(100000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG");
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().Strangle(30, 5, -10));
}
```

   The `Strangle` filter narrows the universe down to just the two contracts you need to form a long strangle.

2. In the `OnData` method, select the contracts of the strategy legs.

```
C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Find options with the farthest expiry
    var expiry = chain.Max(contract => contract.Expiry);
    var contracts = chain.Where(contract => contract.Expiry == expiry).ToList();

    // Order the OTM calls by strike to find the nearest to ATM
    var callContracts = contracts
        .Where(contract => contract.Right == OptionRight.Call &&
            contract.Strike > chain.Underlying.Price)
        .OrderBy(contract => contract.Strike).ToArray();
    if (callContracts.Length == 0) return;

    // Order the OTM puts by strike to find the nearest to ATM
    var putContracts = contracts
        .Where(contract => contract.Right == OptionRight.Put &&
            contract.Strike < chain.Underlying.Price)
        .OrderByDescending(contract => contract.Strike).ToArray();
    if (putContracts.Length == 0) return;

    var call = callContracts[0];
    var put = putContracts[0];
```

3. In the `OnData` method, place the orders.

   Approach A: Call the `OptionStrategies.Strangle` method with the details of each leg and then pass the result to the `Buy` method.

```
C#
var longStrangle = OptionStrategies.Strangle(_symbol, call.Strike, put.Strike, expiry);
Buy(longStrangle, 1);
```

   Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

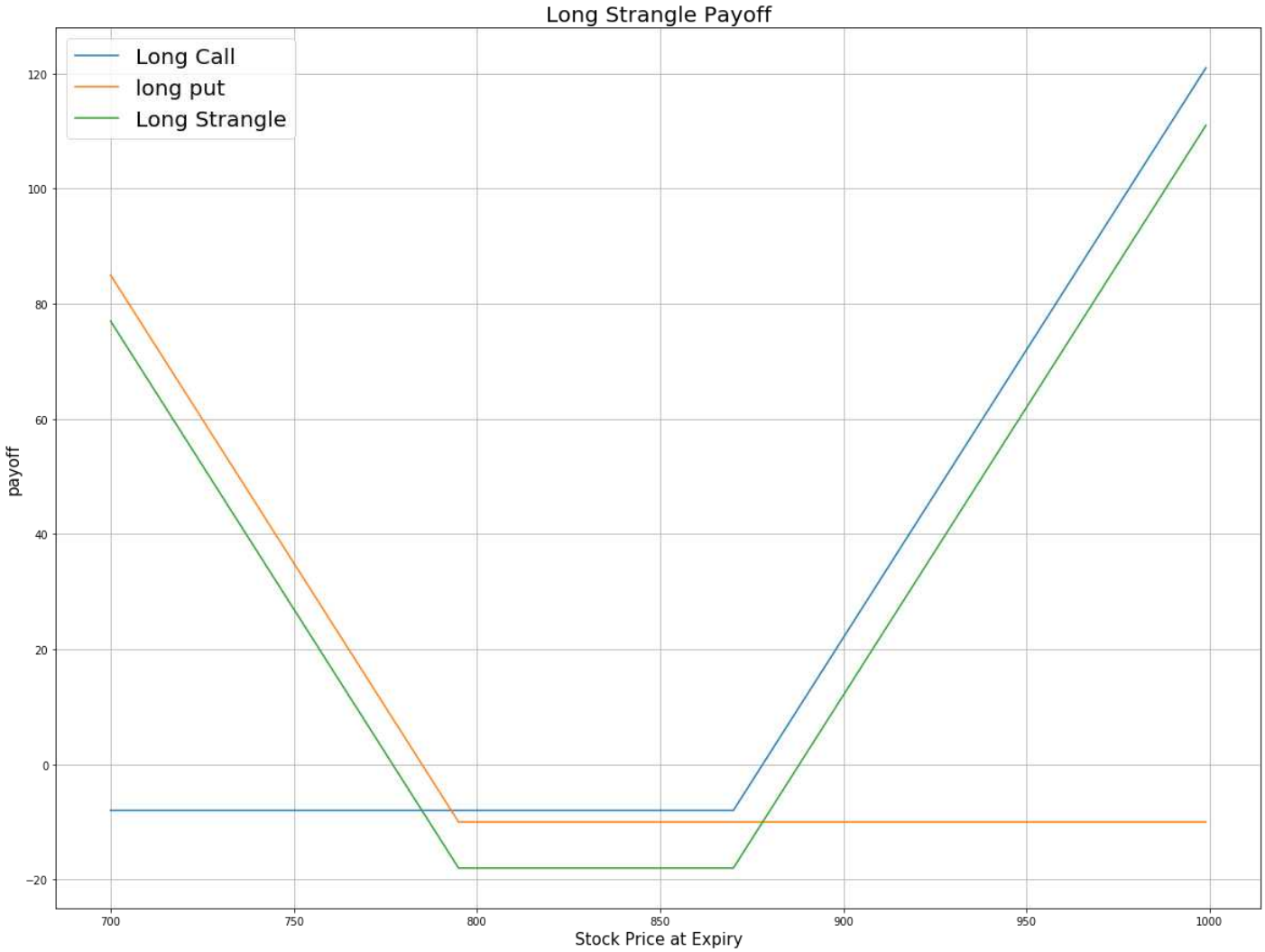
```
C#
var legs = new List<Leg>()
    {
        Leg.Create(call.Symbol, 1),
        Leg.Create(put.Symbol, 1)
    };
ComboMarketOrder(legs, 1);
```

Strategy Payoff

The payoff of the strategy is

$$\begin{array}{rcll} C^{OTM}_T & = & (S_T - K^{C})^{+}\\ P^{OTM}_T & = & (K^{P} - S_T)^{+}\\ P_T & = & (C^{OTM}_T + P^{OTM}_T - C^{OTM}_0 - P^{OTM}_0)\times m - fee \end{array} \begin{array}{rcll} \textrm{where} & C^{OTM}_T & = & \textrm{OTM call value at time T}\\ & P^{OTM}_T & = & \textrm{OTM put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^{C} & = & \textrm{OTM call strike price}\\ & K^{P} & = & \textrm{OTM put strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & C^{OTM}_0 & = & \textrm{OTM call value at position opening (debit paid)}\\ & P^{OTM}_0 & = & \textrm{OTM put value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is unlimited if the underlying price rises to infinity at expiration.

The maximum loss is the net debit paid, C^{OTM}_0 + P^{OTM}_0. It occurs when the underlying price at expiration is the same as when you opened the trade. In this case, both Options expire worthless.

Example

The following table shows the price details of the assets in the algorithm at Option expiration (04/22/2017):

Asset	**Price ($)**	**Strike ($)**
Call	8.80	835.00
Put	9.50	832.50
Underlying Equity at expiration	843.19	-

Therefore, the payoff is

\begin{array}{rcll} C^{OTM}_T & = & (S_T - K^{C})^{+}\\ & = & (843.19-835.00)^{+}\\ & = & 8.19\\ P^{OTM}_T & = & (K^{P} - S_T)^{+}\\ & = & (832.50-843.19)^{+}\\ & = & 0\\ P_T & = & (C^{OTM}_T + P^{OTM}_T - C^{OTM}_0 - P^{OTM}_0)\times m - fee\\ & = & (8.19+0-8.80-9.50)\times100-2.00\times2\\ & = & -1013 \end{array}

So, the strategy loses $1,013.

The following algorithm implements a long strangle Option strategy:

C#

```
public class LongStrangleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("GOOG");
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().Strangle(30, 5, -10));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested ||
            !slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            return;
        }

        // Find options with the farthest expiry
        var expiry = chain.Max(contract => contract.Expiry);
        var contracts = chain.Where(contract => contract.Expiry == expiry).ToList();

        // Order the OTM calls by strike to find the nearest to ATM
        var callContracts = contracts
            .Where(contract => contract.Right == OptionRight.Call &&
                contract.Strike > chain.Underlying.Price)
            .OrderBy(contract => contract.Strike).ToArray();
        if (callContracts.Length == 0) return;

        // Order the OTM puts by strike to find the nearest to ATM
        var putContracts = contracts
            .Where(contract => contract.Right == OptionRight.Put &&
                contract.Strike < chain.Underlying.Price)
            .OrderByDescending(contract => contract.Strike).ToArray();
        if (putContracts.Length == 0) return;

        var callStrike = callContracts[0].Strike;
        var putStrike = putContracts[0].Strike;

        var longStrangle = OptionStrategies.Strangle(_symbol, callStrike, putStrike, expiry);
        Buy(longStrangle, 1);
    }
}
```

Option Strategies

Short Strangle

Introduction

Short Strangle is an Options trading strategy that consists of simultaneously selling an OTM put and an OTM call, where both contracts have the same underlying asset and expiration date. By doing so, the trader is essentially betting that the underlying asset will remain relatively stable and not experience significant price movements before the Options' expiration.

Compared to a short straddle , the net debit of a short strangle is lower since OTM Options are cheaper. Additionally, the winning range of a short straddle is wider and the strike spread is wider.

Implementation

Follow these steps to implement the short strangle strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(100000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG");
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().Strangle(30, 5, -10));
}
```

   The `Strangle` filter narrows the universe down to just the two contracts you need to form a short strangle.

2. In the `OnData` method, select the contracts of the strategy legs.

```
C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Find options with the farthest expiry
    var expiry = chain.Max(contract => contract.Expiry);
    var contracts = chain.Where(contract => contract.Expiry == expiry).ToList();

    // Order the OTM calls by strike to find the nearest to ATM
    var callContracts = contracts
        .Where(contract => contract.Right == OptionRight.Call &&
            contract.Strike > chain.Underlying.Price)
        .OrderBy(contract => contract.Strike).ToArray();
    if (callContracts.Length == 0) return;

    // Order the OTM puts by strike to find the nearest to ATM
    var putContracts = contracts
        .Where(contract => contract.Right == OptionRight.Put &&
            contract.Strike < chain.Underlying.Price)
        .OrderByDescending(contract => contract.Strike).ToArray();
    if (putContracts.Length == 0) return;

    var call = callContracts[0];
    var put = putContracts[0];
```

3. In the `OnData` method, place the orders.

   Approach A: Call the `OptionStrategies.ShortStrangle` method with the details of each leg and then pass the result to the `Buy` method.

```
C#
var shortStrangle = OptionStrategies.ShortStrangle(_symbol, call.Strike, put.Strike, expiry);
Buy(shortStrangle, 1);
```

   Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

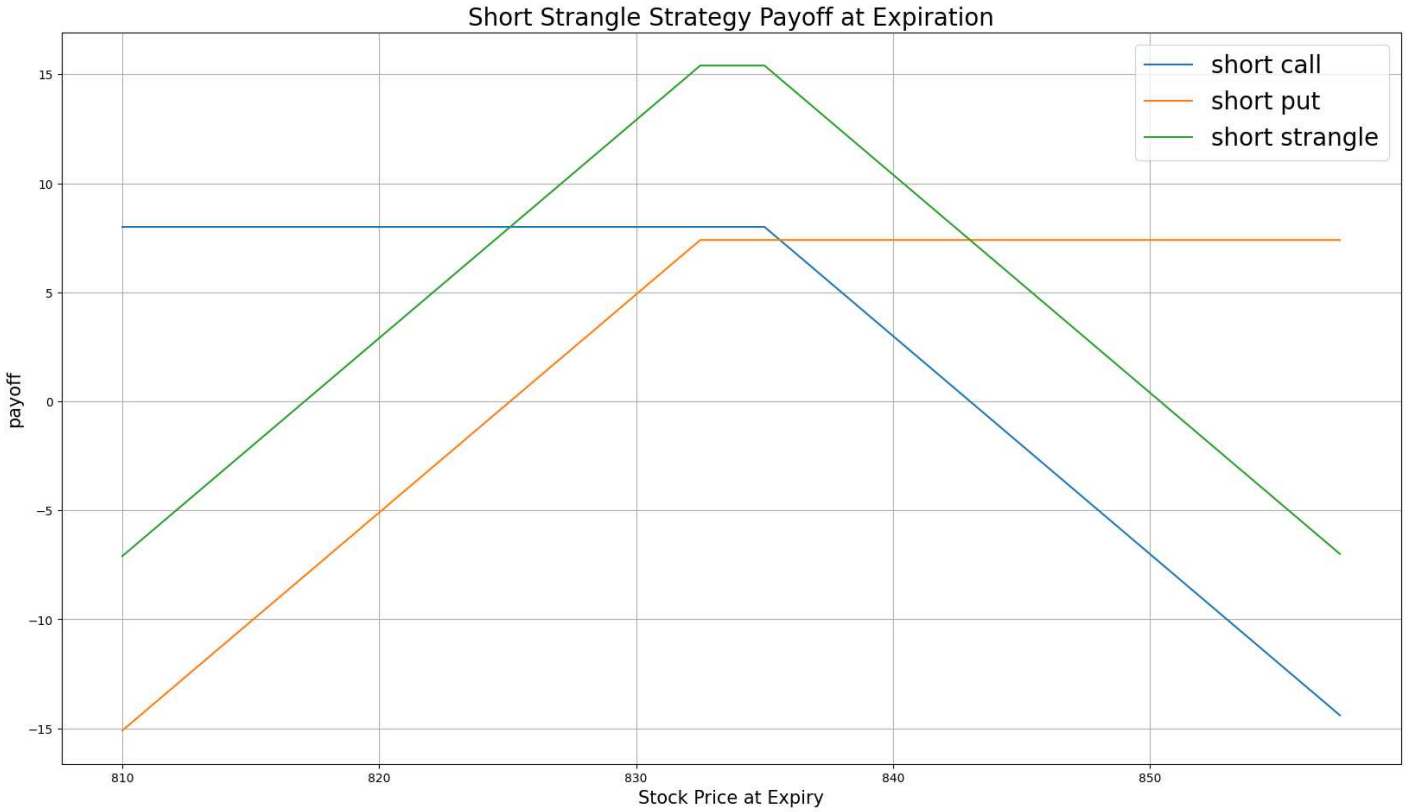
```
C#
var legs = new List<Leg>()
    {
        Leg.Create(call.Symbol, -1),
        Leg.Create(put.Symbol, -1)
    };
ComboMarketOrder(legs, 1);
```

Strategy Payoff

The payoff of the strategy is

$$\begin{array}{rcll} C^{OTM}_T & = & (S_T - K^{C})^{+}\\ P^{OTM}_T & = & (K^{P} - S_T)^{+}\\ P_T & = & (-C^{OTM}_T - P^{OTM}_T + C^{OTM}_0 + P^{OTM}_0)\times m - fee \end{array} \begin{array}{rcl} \textrm{where} & C^{OTM}_T & = & \textrm{OTM call value at time T}\\ & P^{OTM}_T & = & \textrm{OTM put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^{C} & = & \textrm{OTM call strike price}\\ & K^{P} & = & \textrm{OTM put strike price}\\ & P_T & = & \textrm{Payout total at time T}\\ & C^{OTM}_0 & = & \textrm{OTM call value at position opening (debit paid)}\\ & P^{OTM}_0 & = & \textrm{OTM put value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit C^{OTM}_0 + P^{OTM}_0. It occurs when the underlying price at expiration remains within the range of the strike prices. In this case, both Options expire worthless.

The maximum loss is unlimited if the underlying price rises to infinity or substantial, C^{OTM}_0 + P^{OTM}_0 - K^{P}, if it drops to zero at expiration.

If the Option is American Option, there is a risk of early assignment on the contracts you sell.

Example

The following table shows the price details of the assets in the algorithm at Option expiration (2017-04-22):

Asset	**Price ($)**	**Strike ($)**
Call	8.00	835.00
Put	7.40	832.50
Underlying Equity at expiration	843.19	-

Therefore, the payoff is

\begin{array}{rcll} C^{OTM}_T & = & (S_T - K^{C})^{+}\\ & = & (843.19-835.00)^{+}\\ & = & 8.19\\ P^{OTM}_T & = & (K^{P} - S_T)^{+}\\ & = & (832.50-843.19)^{+}\\ & = & 0\\ P_T & = & (-C^{OTM}_T - P^{OTM}_T + C^{OTM}_0 + P^{OTM}_0)\times m - fee\\ & = & (-8.19-0+8.00+7.40)\times100-2.00\times2\\ & = & 719 \end{array}

So, the strategy gains $719.

The following algorithm implements a short strangle Option strategy:

C#

```
public class ShortStrangleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("GOOG");
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().Strangle(30, 5, -10));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested ||
            !slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            return;
        }

        // Find options with the farthest expiry
        var expiry = chain.Max(contract => contract.Expiry);
        var contracts = chain.Where(contract => contract.Expiry == expiry).ToList();

        // Order the OTM calls by strike to find the nearest to ATM
        var callContracts = contracts
            .Where(contract => contract.Right == OptionRight.Call &&
                contract.Strike > chain.Underlying.Price)
            .OrderBy(contract => contract.Strike).ToArray();
        if (callContracts.Length == 0) return;

        // Order the OTM puts by strike to find the nearest to ATM
        var putContracts = contracts
            .Where(contract => contract.Right == OptionRight.Put &&
                contract.Strike < chain.Underlying.Price)
            .OrderByDescending(contract => contract.Strike).ToArray();
        if (putContracts.Length == 0) return;

        var callStrike = callContracts[0].Strike;
        var putStrike = putContracts[0].Strike;

        var shortStrangle = OptionStrategies.ShortStrangle(_symbol, callStrike, putStrike, expiry);
        Buy(shortStrangle, 1);
    }
}
```

Option Strategies

Conversion

Introduction

A Conversion is a special case of a protective collar . It consist of holding one lot of the underlying security, shorting a call, and longing a put with lower strike price. However, the strategy now serves as an delta-neutral arbitration from Option mispricing instead of a hedge strategy. Note that it only attains a true profit when the risk-free return is greater than the risk-free interest rate .

Implementation

Follow these steps to implement the conversion strategy:

1. In the `Initialize` method, set the start date, set the end date, subscribe to the underlying Equity , and create an Option universe .

   ```C#
   private Symbol _symbol;

   public override void Initialize()
   {
       SetStartDate(2024, 9, 1);
       SetEndDate(2024, 12, 31);
       SetCash(100000);

       UniverseSettings.Asynchronous = true;
       var option = AddOption("GOOG", Resolution.Minute);
       _symbol = option.Symbol;
       option.SetFilter(universe => universe.IncludeWeeklys().Conversion(30, -5));
   }
   ```

   The `Conversion` filter narrows the universe down to just the two contracts you need to form a conversion.

2. In the `OnData` method, select the strike and expiry of the contracts in the strategy legs.

   ```C#
   public override void OnData(Slice slice)
   {
       if (Portfolio.Invested) return;

       // Get the OptionChain
       if (!slice.OptionChains.TryGetValue(_symbol, out var chain)) return;

       // Select an expiry date and ATM strike price
       var expiry = chain.Max(x => x.Expiry);
       var strike = chain.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First().Strike;
   ```

3. In the `OnData` method, select the contracts and place the orders.

   Approach A: Call the `OptionStrategies.Conversion` method with the details of each leg and then pass the result to the `Buy` method.

   ```C#
   var conversion = OptionStrategies.Conversion(_symbol, strike, expiry);
   Buy(conversion, 1);
   ```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

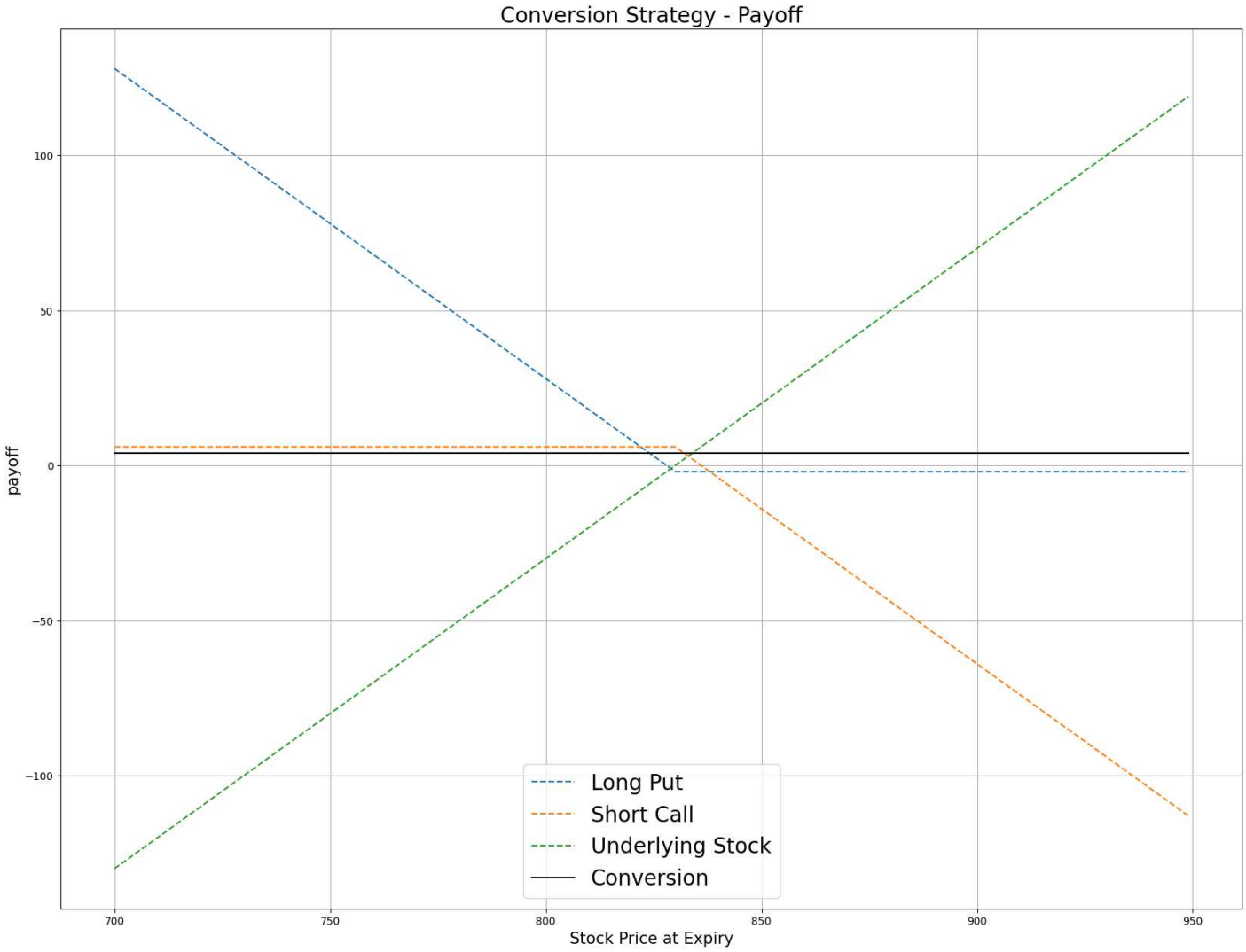
```
C#
// Select the call and put contracts
var call = chain.Single(x => x.Expiry == expiry && x.Strike == strike && x.Right == OptionRight.Call);
var put = chain.Single(x => x.Expiry == expiry && x.Strike == strike && x.Right == OptionRight.Put);

var legs = new List<Leg>()
    {
        Leg.Create(call.Symbol, -1),
        Leg.Create(put.Symbol, 1),
        Leg.Create(chain.Underlying.Symbol, chain.Underlying.SymbolProperties.ContractMultiplier)
    };
ComboMarketOrder(legs, 1);
```

Strategy Payoff

This is a fixed payoff, delta-neutral strategy. The payoff is

$$\begin{array}{rcll} C_T & = & (S_T - K)^{+}\\ P_T & = & (K - S_T)^{+}\\ Payoff_T & = & (S_T - S_0 - C_T + P_T + C_0 - P_0)\times m - fee\\ & = & (K - S_0 + C_0 - P_0)\times m - fee \end{array} \begin{array}{rcll} \textrm{where} & C_T & = & \textrm{Call value at time T}\\ & P_T & = & \textrm{Put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K & = & \textrm{Strike price}\\ & Payoff_T & = & \textrm{Payout total at time T}\\ & S_0 & = & \textrm{Underlying asset price when the trade opened}\\ & C_0 & = & \textrm{Call price when the trade opened (credit received)}\\ & P_0 & = & \textrm{Put price when the trade opened (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The payoff is only dependent on the strike price and the initial asset prices.

If the Option is American Option, there is a risk of early assignment on the contract you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	Price ($)	Strike ($)
Call	8.10	832.50
Put	9.50	832.50
Underlying Equity at position opens	833.17	-

Therefore, the payoff is

$$\begin{array}{rcll} Payoff_T & = & (K - S_0 + C_0 - P_0)\times m - fee\\ & = & (832.50 - 833.17 + 8.10 - 9.50)\times100-1.00\times3\\ & = & -210.00\\ \end{array}$$

So, the strategy loses $210.

The following algorithm implements a conversion Option strategy:

```C#
public class ConversionStrategy : QCAlgorithm
{
    private Symbol _equity;
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        _equity = AddEquity("GOOG", Resolution.Minute).Symbol;
        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().Conversion(30, -5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested) return;

        // Get the OptionChain of the symbol
        if (!slice.OptionChains.TryGetValue(_symbol, out var chain)) return;

        // sort the optionchain by expiration date and choose the furthest date
        var expiry = chain.OrderBy(x => x.Expiry).Last().Expiry;

        // select ATM strike price
        var strike = chain.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First().Strike;

        // Order Strategy
        var conversion = OptionStrategies.Conversion(_symbol, strike, expiry);
        Buy(conversion, 1);
    }

    public override void OnEndOfDay(Symbol symbol)
    {
        if (symbol.Value == "GOOG")
        {
            Log($"{Time}::{symbol}::{Securities[symbol].Price}");
        }
    }
}
```

Option Strategies

Reverse Conversion

Introduction

A Reverse Conversion , or Reversal , is the inverse of a conversion and also a special case of a protective collar . It consist of shorting one lot of the underlying security, buying a call, and selling a put with lower strike price. However, the strategy now serves as an delta-neutral arbitration from Option mispricing instead of a hedge strategy. Note that it only attains a true profit when the risk-free return is greater than the risk-free interest rate .

Implementation

Follow these steps to implement the reverse conversion strategy:

1. In the `Initialize` method, set the start date, set the end date, subscribe to the underlying Equity , and create an Option universe .

   ```C#
   private Symbol _symbol;

   public override void Initialize()
   {
       SetStartDate(2024, 9, 1);
       SetEndDate(2024, 12, 31);
       SetCash(100000);

       UniverseSettings.Asynchronous = true;
       var option = AddOption("GOOG", Resolution.Minute);
       _symbol = option.Symbol;
       option.SetFilter(universe => universe.IncludeWeeklys().Conversion(30, -5));
   }
   ```

   The `Conversion` filter narrows the universe down to just the two contracts you need to form a reverse conversion.

2. In the `OnData` method, select the strike and expiry of the contracts in the strategy legs.

   ```C#
   public override void OnData(Slice slice)
   {
       if (Portfolio.Invested) return;

       // Get the OptionChain
       if (!slice.OptionChains.TryGetValue(_symbol, out var chain)) return;

       // Select an expiry date and ATM strike price
       var expiry = chain.Max(x => x.Expiry);
       var strike = chain.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First().Strike;
   ```

3. In the `OnData` method, select the contracts and place the orders.

   Approach A: Call the `OptionStrategies.ReverseConversion` method with the details of each leg and then pass the result to the `Buy` method.

   ```C#
   var reverseConversion = OptionStrategies.ReverseConversion(_symbol, strike, expiry);
   Buy(reverseConversion, 1);
   ```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

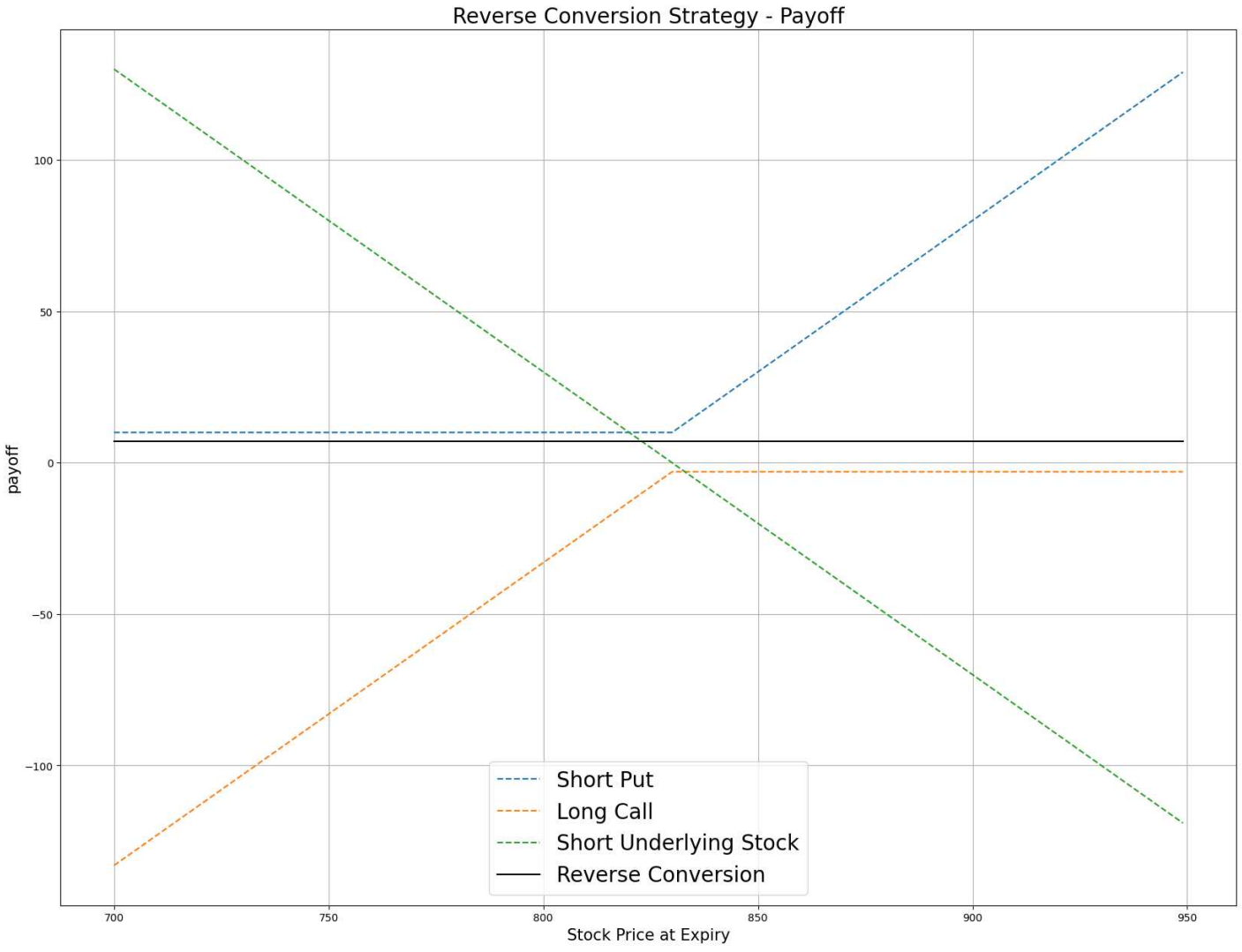
```
C#
// Select the call and put contracts
var call = chain.Single(x => x.Expiry == expiry && x.Strike == strike && x.Right == OptionRight.Call);
var put = chain.Single(x => x.Expiry == expiry && x.Strike == strike && x.Right == OptionRight.Put);

var legs = new List<Leg>()
    {
        Leg.Create(call.Symbol, 1),
        Leg.Create(put.Symbol, -1),
        Leg.Create(chain.Underlying.Symbol, -chain.Underlying.SymbolProperties.ContractMultiplier)
    };
ComboMarketOrder(legs, 1);
```

Strategy Payoff

This is a fixed payoff, delta-neutral strategy. The payoff is

$$\begin{array}{rcll} C_T & = & (S_T - K)^{+}\\ P_T & = & (K - S_T)^{+}\\ Payoff_T & = & (S_0 - S_T + C_T - P_T - C_0 + P_0)\times m - fee\\ & = & (S_0 - K - C_0 + P_0)\times m - fee \end{array} \begin{array}{rcl} \textrm{where} & C_T & = & \textrm{Call value at time T}\\ & P_T & = & \textrm{Put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K & = & \textrm{Strike price}\\ & Payoff_T & = & \textrm{Payout total at time T}\\ & S_0 & = & \textrm{Underlying asset price when the trade opened}\\ & C_0 & = & \textrm{Call price when the trade opened (credit received)}\\ & P_0 & = & \textrm{Put price when the trade opened (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The payoff is only dependent on the strike price and the initial asset prices.

If the Option is American Option, there is a risk of early assignment on the contract you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	Price ($)	Strike ($)
Call	10.30	832.50
Put	7.40	832.50
Underlying Equity at position opens	832.26	-

Therefore, the payoff is

$$\begin{array}{rcll} Payoff_T & = & (S_0 - K - C_0 + P_0)\times m - fee\\ & = & (832.26 - 832.50 - 10.30 + 7.40)\times100-1.00\times3\\ & = & -317.00\\ \end{array}$$

So, the strategy loses $317.

The following algorithm implements a reverse conversion Option strategy:

```C#
public class ReverseConversionStrategy : QCAlgorithm
{
    private Symbol _equity;
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        _equity = AddEquity("GOOG", Resolution.Minute).Symbol;
        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().Conversion(30, -5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested) return;

        // Get the OptionChain of the symbol
        if (!slice.OptionChains.TryGetValue(_symbol, out var chain)) return;

        // sort the optionchain by expiration date and choose the furthest date
        var expiry = chain.OrderBy(x => x.Expiry).Last().Expiry;

        // select ATM strike price
        var strike = chain.OrderBy(x => Math.Abs(x.Strike - chain.Underlying.Price)).First().Strike;

        // Order Strategy
        var reverseConversion = OptionStrategies.ReverseConversion(_symbol, strike, expiry);
        Buy(reverseConversion, 1);
    }

    public override void OnEndOfDay(Symbol symbol)
    {
        if (symbol.Value == "GOOG")
        {
            Log($"{Time}::{symbol}::{Securities[symbol].Price}");
        }
    }
}
```

Option Strategies

Long Box Spread

Introduction

A long Box Spread is the combination of a bull call spread and a bear put spread . It consist of buying an ITM call at strike A, selling an OTM put at strike A, buying an ITM put at strike B < A, and selling an OTM call at strike B, where all of the contracts have the same expiry date. This strategy serves as an delta-neutral arbitration from Option mispricing. Note that it only attains a true profit when the risk-free return is greater than the risk-free interest rate .

Implementation

Follow these steps to implement the long box spread strategy:

1. In the `Initialize` method, set the start date, set the end date, subscribe to the underlying Equity , and create an Option universe .

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(100000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().BoxSpread(30, 5));
}
```

   The `BoxSpread` filter narrows the universe down to just the four contracts you need to form a box spread.

2. In the `OnData` method, select the expiry and strikes of the contracts in the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested) return;

    // Get the OptionChain
    if (!slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select an expiry date and ITM & OTM strike prices
    var expiry = chain.Max(x => x.Expiry);
    var contracts = chain.Where(x => x.Expiry == expiry).ToList();
    var higherStrike = contracts.Max(x => x.Strike);
    var lowerStrike = contracts.Min(x => x.Strike);
```

3. In the `OnData` method, select the contracts and place the order.

   Approach A: Call the `OptionStrategies.BoxSpread` method with the details of each leg and then pass the result to the `Buy` method.

```C#
var boxSpread = OptionStrategies.BoxSpread(_symbol, higherStrike, lowerStrike, expiry);
Buy(boxSpread, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

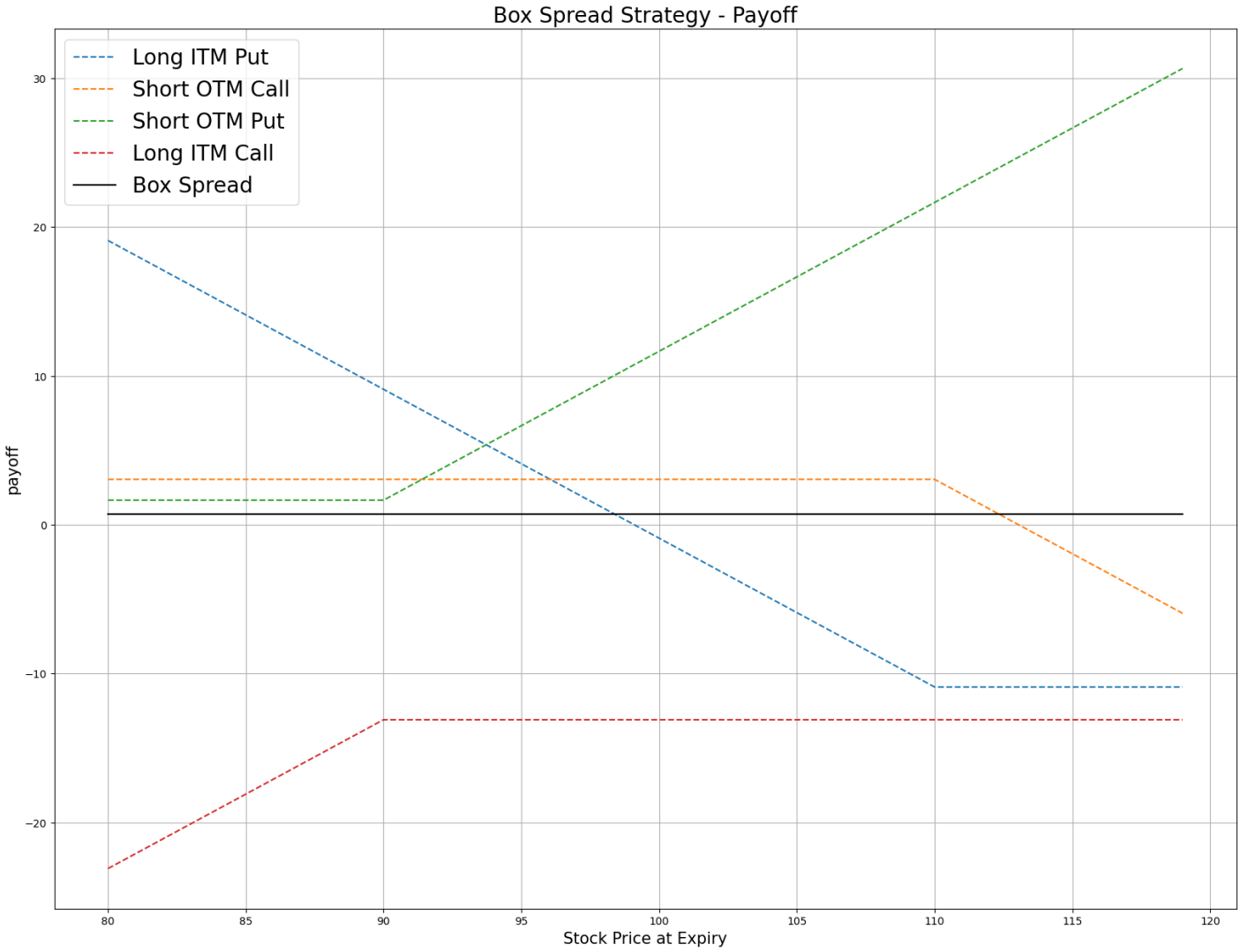
```C#
// Select the call and put contracts
var itmCall = contracts.Single(x => x.Strike == lowerStrike && x.Right == OptionRight.Call);
var otmCall = contracts.Single(x => x.Strike == higherStrike && x.Right == OptionRight.Call);
var itmPut = contracts.Single(x => x.Strike == higherStrike && x.Right == OptionRight.Put);
var otmPut = contracts.Single(x => x.Strike == lowerStrike && x.Right == OptionRight.Put);

var legs = new List<Leg>()
{
    Leg.Create(itmCall.Symbol, 1),
    Leg.Create(itmPut.Symbol, 1),
    Leg.Create(otmCall.Symbol, -1),
    Leg.Create(otmPut.Symbol, -1),
};
ComboMarketOrder(legs, 1);
```

Strategy Payoff

This is a fixed payoff, delta-neutral strategy. The payoff is

$$\begin{array}{rcll} C_T^{ITM} & = & (S_T - K_{-})^{+}\\ C_T^{OTM} & = & (S_T - K_{+})^{+}\\ P_T^{ITM} & = & (K_{+} - S_T)^{+}\\ P_T^{OTM} & = & (K_{-} - S_T)^{+}\\ Payoff_T & = & (C_T^{ITM} - C_{T_0}^{ITM} + P_T^{ITM} - P_{T_0}^{ITM} - C_T^{OTM} + C_{T_0}^{OTM} - P_T^{OTM} + P_{T_0}^{OTM})\times m - fee\\ & = & (K_{+} - K_{-} + C_{T_0}^{OTM} + P_{T_0}^{OTM} - C_{T_0}^{ITM} - P_{T_0}^{ITM})\times m - fee \end{array} \begin{array}{rcll} \textrm{where} & C_T^{ITM} & = & \textrm{ITM Call value at time T}\\ & C_T^{OTM} & = & \textrm{OTM Call value at time T}\\ & P_T^{ITM} & = & \textrm{ITM Put value at time T}\\ & P_T^{OTM} & = & \textrm{OTM Put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K_{+} & = & \textrm{Higher strike price}\\ & K_{-} & = & \textrm{Lower strike price}\\ & Payoff_{T_1} & = & \textrm{Payout total at time T1}\\ & C_{T_0}^{ITM} & = & \textrm{ITM Call price when the trade opened (debit paid)}\\ & C_{T_0}^{OTM} & = & \textrm{OTM Call price when the trade opened (credit received)}\\ & P_{T_0}^{ITM} & = & \textrm{ITM Put price when the trade opened (debit paid)}\\ & P_{T_0}^{OTM} & = & \textrm{OTM Put price when the trade opened (credit received)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The payoff is only dependent on the strike price and the initial asset prices.

If the Option is American Option, there is a risk of early assignment on the contracts you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	**Price ($)**	**Strike ($)**
ITM Call	27.30	810.00
ITM Put	28.00	857.50
OTM Call	1.05	857.50
OTM Put	1.50	810.00
Underlying Equity at expiration	843.25	-

Therefore, the payoff is

$$\begin{array}{rcll} Payoff_T & = & (K_{+} - K_{-} + C_0^{OTM} + P_0^{OTM} - C_0^{ITM} - P_0^{ITM})\times m - fee\\ & = & (857.50 - 810.00 + 1.05 + 1.50 - 27.30 - 28.00)\times100 - 1.00\times4\\ & = & -529.00\\ \end{array}$$

So, the strategy loses $529.

The following algorithm implements a long box spread Option strategy:

```C#
public class BoxSpreadStrategy : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        UniverseSettings.Asynchronous = true;

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().BoxSpread(30, 5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested) return;

        // Get the OptionChain
        if (!slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            return;
        }

        // Select an expiry date
        var expiry = chain.Max(x => x.Expiry);

        // Select the strike prices of the contracts
        var contracts = chain.Where(x => x.Expiry == expiry).ToList();
        var higherStrike = contracts.Max(x => x.Strike);
        var lowerStrike = contracts.Min(x => x.Strike);
        var boxSpread = OptionStrategies.BoxSpread(_symbol, higherStrike, lowerStrike, expiry);
        Buy(boxSpread, 1);
    }

    public override void OnEndOfDay(Symbol symbol)
    {
        if (symbol == _symbol.Underlying)
        {
            Log($"{Time}::{symbol}::{Securities[symbol].Price}");
        }
    }
}
```

Option Strategies

Short Box Spread

Introduction

A Short Box Spread is the inverse of a box spread , as well as the combination of a bear call spread and a bull put spread . It consists of buying an OTM call at strike A, selling an ITM put at strike A, buying an OTM put and strike B < A, and selling an ITM call at strike B, where all of the contracts have the same expiry date. This strategy serves as an delta-neutral arbitration from Option mispricing. Note that it only attains a true profit when the risk-free return is greater than the risk-free interest rate .

Implementation

Follow these steps to implement the short box spread strategy:

1. In the `Initialize` method, set the start date, set the end date, subscribe to the underlying Equity , and create an Option universe .

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(100000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().BoxSpread(30, 5));
}
```

   The `BoxSpread` filter narrows the universe down to just the four contracts you need to form a short box spread.

2. In the `OnData` method, select the strike and expiry of the contracts in the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested) return;

    // Get the OptionChain
    if (!slice.OptionChains.TryGetValue(_symbol, out var chain)) return;

    // Select an expiry date and ITM & OTM strike prices
    var expiry = chain.Max(x => x.Expiry);
    var contracts = chain.Where(x => x.Expiry == expiry).ToList();
    var higherStrike = contracts.Max(x => x.Strike);
    var lowerStrike = contracts.Min(x => x.Strike);
```

3. In the `OnData` method, select the contracts and place the orders.

   Approach A: Call the `OptionStrategies.ShortBoxSpread` method with the details of each leg and then pass the result to the `Buy` method.

   ```
   C#
   var shortBoxSpread = OptionStrategies.ShortBoxSpread(_symbol, higherStrike, lowerStrike, expiry);
   Buy(shortBoxSpread, 1);
   ```

   Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

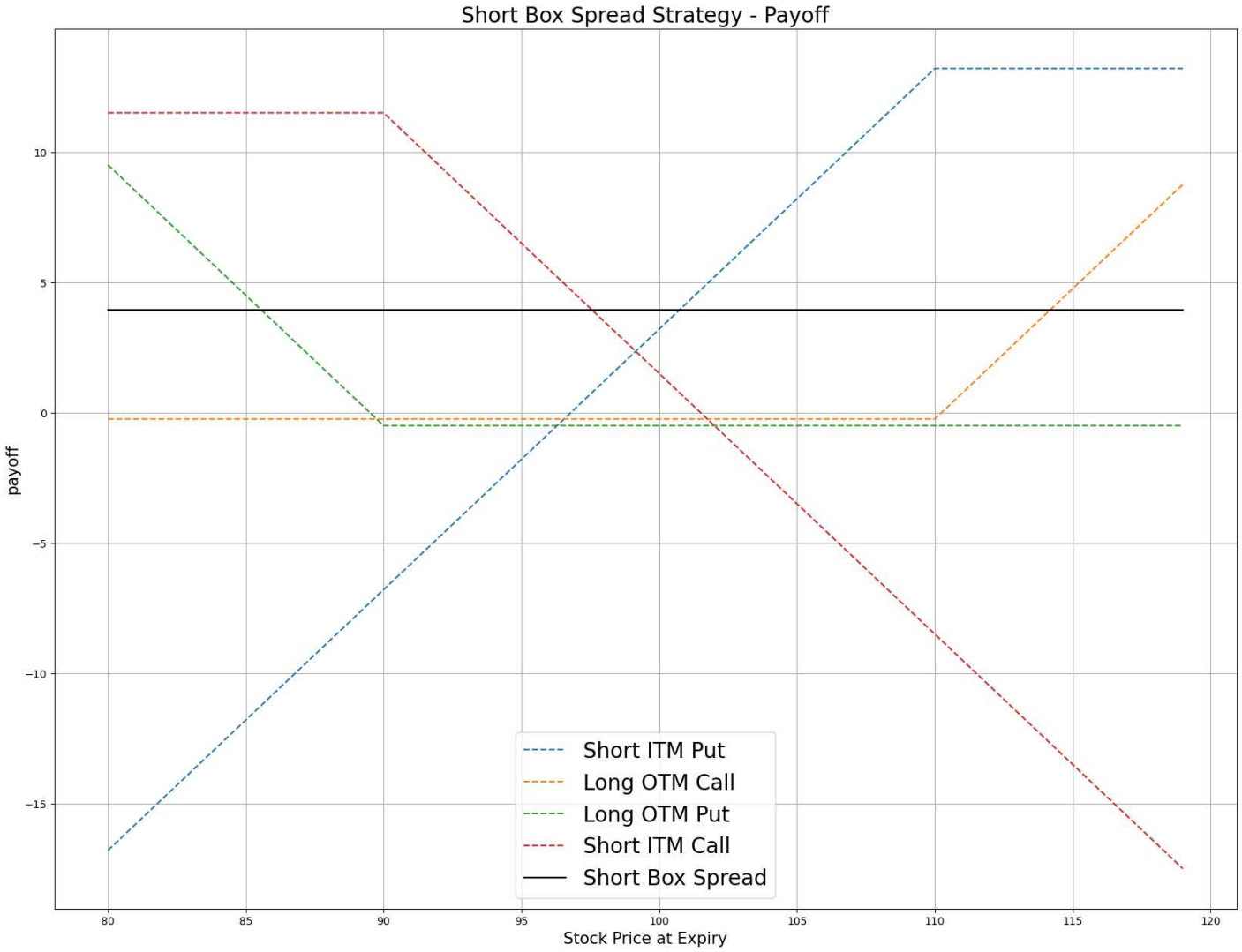
   ```
   C#
   // Select the call and put contracts
   var itmCall = chain.Single(x => x.Expiry == expiry && x.Strike == lowerStrike && x.Right == OptionRight.Call);
   var otmCall = chain.Single(x => x.Expiry == expiry && x.Strike == higherStrike && x.Right == OptionRight.Call);
   var itmPut = chain.Single(x => x.Expiry == expiry && x.Strike == higherStrike && x.Right == OptionRight.Put);
   var otmPut = chain.Single(x => x.Expiry == expiry && x.Strike == lowerStrike && x.Right == OptionRight.Put);

   var legs = new List<Leg>()
       {
           Leg.Create(itmCall.Symbol, -1),
           Leg.Create(itmPut.Symbol, -1),
           Leg.Create(otmCall.Symbol, 1),
           Leg.Create(otmPut.Symbol, 1),
       };
   ComboMarketOrder(legs, 1);
   ```

Strategy Payoff

This is a fixed payoff, delta-neutral strategy. The payoff is

$$\begin{array}{rcll} C_T^{ITM} & = & (S_T - K_{-})^{+}\\ C_T^{OTM} & = & (S_T - K_{+})^{+}\\ P_T^{ITM} & = & (K_{+} - S_T)^{+}\\ P_T^{OTM} & = & (K_{-} - S_T)^{+}\\ Payoff_T & = & (C_{T_0}^{ITM} - C_T^{ITM} + P_{T_0}^{ITM} - P_T^{ITM} - C_{T_0}^{OTM} + C_T^{OTM} - P_{T_0}^{OTM} + P_T^{OTM})\times m - fee\\ & = & (K_{-} - K_{+} + C_{T_0}^{ITM} + P_{T_0}^{ITM} - C_{T_0}^{OTM} - P_{T_0}^{OTM})\times m - fee \end{array} \begin{array}{rcll} \textrm{where} & C_T^{ITM} & = & \textrm{ITM Call value at time T}\\ & C_T^{OTM} & = & \textrm{OTM Call value at time T}\\ & P_T^{ITM} & = & \textrm{ITM Put value at time T}\\ & P_T^{OTM} & = & \textrm{OTM Put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K_{+} & = & \textrm{Higher strike price}\\ & K_{-} & = & \textrm{Lower strike price}\\ & Payoff_T & = & \textrm{Payout total at time T}\\ & C_{T_0}^{ITM} & = & \textrm{ITM Call price when the trade opened (debit paid)}\\ & C_{T_0}^{OTM} & = & \textrm{OTM Call price when the trade opened (credit received)}\\ & P_{T_0}^{ITM} & = & \textrm{ITM Put price when the trade opened (debit paid)}\\ & P_{T_0}^{OTM} & = & \textrm{OTM Put price when the trade opened (credit received)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The payoff is only dependent on the strike price and the initial asset prices.

If the Option is American Option, there is a risk of early assignment on the contracts you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	**Price ($)**	**Strike ($)**
ITM Call	23.00	810.00
ITM Put	23.80	857.50
OTM Call	1.85	857.50
OTM Put	2.75	810.00
Underlying Equity at expiration	843.25	-

Therefore, the payoff is

$$\begin{array}{rcll} Payoff_T & = & (K_{-} - K_{+} + C_0^{ITM} + P_0^{ITM} - C_0^{OTM} - P_0^{OTM})\times m - fee\\ & = & (810.00 - 857.50 + 23.00 + 23.80 - 1.85 - 2.75)\times100 - 1.00\times4\\ & = & -534.00\\ \end{array}$$

So, the strategy loses $534.

The following algorithm implements a short box spread Option strategy:

```C#
public class ProtectiveCollarStrategy : QCAlgorithm
{
    private Symbol _equitySymbol;
    private Symbol _optionSymbol;
    private bool _investEver = false;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        UniverseSettings.Asynchronous = true;
        _equitySymbol = AddEquity("GOOG").Symbol;

        var option = AddOption("GOOG", Resolution.Minute);
        _optionSymbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().BoxSpread(30, 5));
    }

    public override void OnData(Slice slice)
    {
        if (_investEver) return;

        // Get the OptionChain
        var chain = slice.OptionChains.get(_optionSymbol, null);
        if (chain == null || chain.Count() == 0) return;

        // Select an expiry date
        var expiry = chain.OrderBy(x => x.Expiry).Last().Expiry;

        // Select the strike prices of the contracts
        var orderedContracts = chain.OrderBy(x => x.Strike);
        var higherStrike = orderedContracts.Last().Strike;
        var lowerStrike = orderedContracts.First().Strike;
        var boxSpread = OptionStrategies.ShortBoxSpread(_optionSymbol, higherStrike, lowerStrike, expiry);
        Buy(boxSpread, 1);

        _investEver = true;
    }

    public override void OnEndOfDay(Symbol symbol)
    {
        if (symbol == _equitySymbol)
        {
            Log($"{Time}::{symbol}::{Securities[symbol].Price}");
        }
    }
}
```

Option Strategies

Long Jelly Roll

Introduction

A long Jelly Roll , or simply Roll , is a combination of a long call calendar spread and a short put calendar spread . It consists of buying a put and selling a call of the same expiry, as well as buying a call and selling a put with a further expiry, where all of the contracts have the same strike price. This strategy serves as an arbitrage on Option mispricing due to the temporary disparity between the call spread and the put spread synthetic portfolios. It is a delta -, gamma -, vega -, and theta -neutral strategy, but sensitive to rho (interest rate) and phi (dividend yield).

Implementation

Follow these steps to implement the long jelly roll strategy:

1. In the `Initialize` method, set the start date, set the end date, and create an Option universe .

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(100000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(x => x.IncludeWeeklys().JellyRoll(5m, 30, 60));
}
```

2. In the `OnData` method, select the expiry and strikes of the contracts in the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested) return;

    // Get the OptionChain
    if (!slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select expiry dates and strike price
    var strike = chain.OrderBy(x => Math.Abs(chain.Underlying.Price - x.Strike)).First().Strike;
    var contracts = chain.Where(x => x.Strike == strike).ToList();
    var farExpiry = contracts.Max(x => x.Expiry);
    var nearExpiry = contracts.Min(x => x.Expiry);
```

3. In the `OnData` method, select the contracts and place the order.

   Approach A: Call the `OptionStrategies.JellyRoll` method with the details of each leg and then pass the result to the `Buy` method.

```C#
var jellyRoll = OptionStrategies.JellyRoll(_symbol, strike, nearExpiry, farExpiry);
Buy(jellyRoll, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

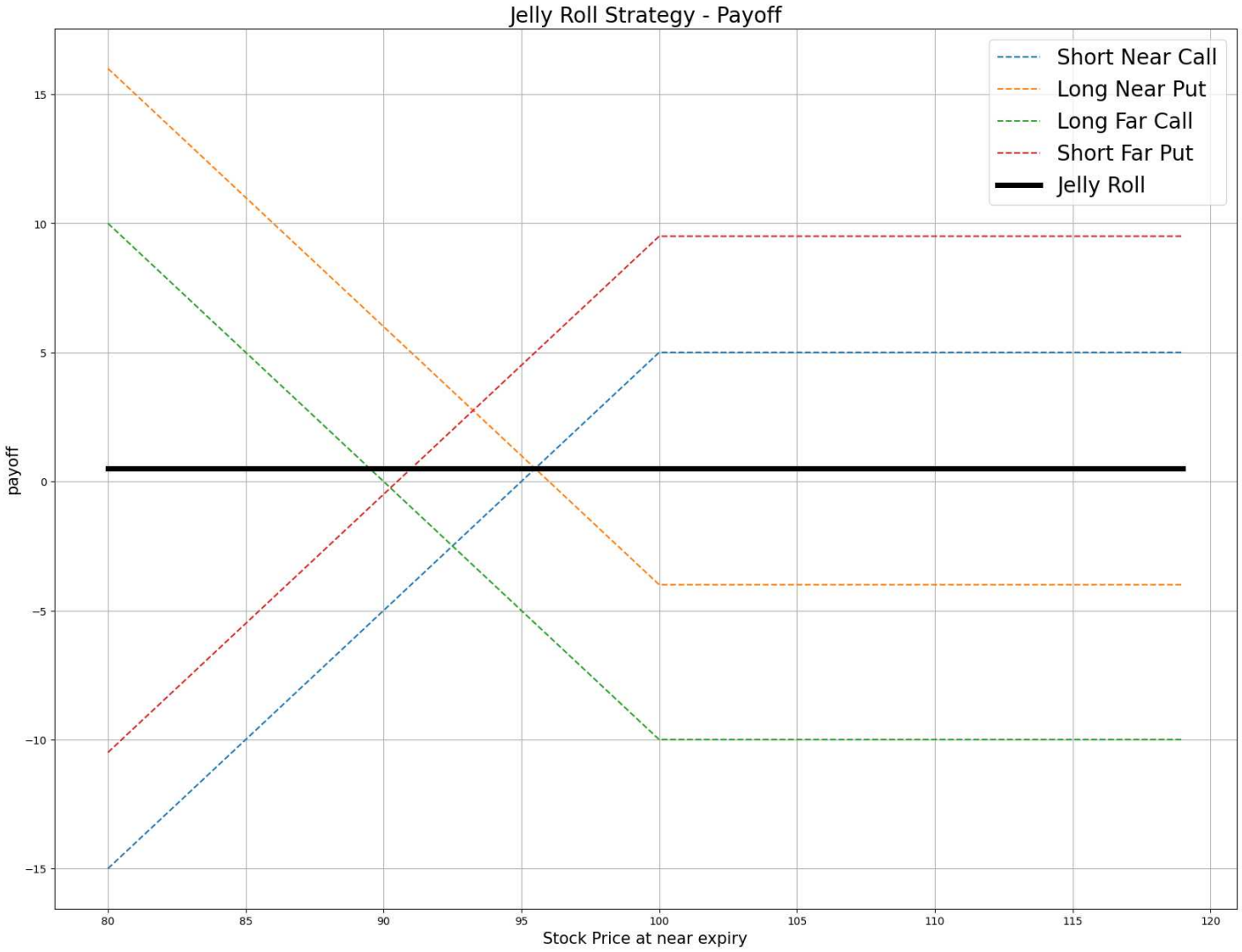
```C#
// Select the call and put contracts
var nearCall = contracts.Single(x => x.Expiry == nearExpiry && x.Right == OptionRight.Call);
var farCall = contracts.Single(x => x.Expiry == farExpiry && x.Right == OptionRight.Call);
var nearPut = contracts.Single(x => x.Expiry == nearExpiry && x.Right == OptionRight.Put);
var farPut = contracts.Single(x => x.Expiry == farExpiry && x.Right == OptionRight.Put);

var legs = new List<Leg>()
{
    Leg.Create(nearCall.Symbol, -1),
    Leg.Create(farCall.Symbol, 1),
    Leg.Create(nearPut.Symbol, 1),
    Leg.Create(farPut.Symbol, -1),
};
ComboMarketOrder(legs, 1);
```

Strategy Payoff

This is a delta-, gamma-, vega-, and theta-neutral strategy. The payoff is

$$\begin{array}{rcll} C_{T_1}^{T_1} & = & (S_{T_1} - K)^{+}\\ P_{T_1}^{T_1} & = & (K - S_{T_1})^{+}\\ Payoff_{T_1} & = & (P_{T_1}^{T_1} - P_{T_0}^{T_1} - C_{T_1}^{T_1} + C_{T_0}^{T_1} + C_{T_1}^{T_2} - C_{T_0}^{T_2} - P_{T_1}^{T_2} + P_{T_0}^{T_2})\times m - fee\\ \end{array} \begin{array}{rcll} \textrm{where} & C_{T_1}^{T_1} & = & \textrm{Market value of Call with expiry at T1 at time T1}\\ & C_{T_1}^{T_2} & = & \textrm{Market value of Call with expiry at T2 at time T1}\\ & P_{T_1}^{T_1} & = & \textrm{Market value of Put with expiry at T1 at time T1}\\ & P_{T_1}^{T_2} & = & \textrm{Market value of Put with expiry at T2 at time T1}\\ & S_{T_1} & = & \textrm{Underlying asset price at time T1}\\ & K & = & \textrm{Strike price}\\ & Payoff_T & = & \textrm{Payout total at time T}\\ & C_{T_0}^{T_1} & = & \textrm{Market value of Call with expiry at T1 when the trade opened}\\ & C_{T_0}^{T_2} & = & \textrm{Market value of Call with expiry at T2 when the trade opened}\\ & P_{T_0}^{T_1} & = & \textrm{Market value of Put with expiry at T2 when the trade opened}\\ & P_{T_0}^{T_2} & = & \textrm{Market value of Put with expiry at T2 when the trade opened}\\ & m & = & \textrm{Contract multiplier}\\ & T_1 & = & \textrm{Time T1 as the near expiration}\\ & T_2 & = & \textrm{Time T2 as the far expiration} \end{array}$$

The following chart shows the payoff at expiration:



The payoff is dependent on the market prices of the Options, but in theory, if assuming call-put parity exists, the expected payoff would be

$$\begin{array}{rcll} Payoff_{T_1} & = & Payoff_{\textrm{call calendar spread}} - Payoff_{\textrm{put calendar spread}}\\ & = & K \times (T_2 - T_1) \times r - D \end{array} \begin{array}{rcl} \textrm{where} & r & = & \textrm{Continuous compounding interest rate}\\ & D & = & \textrm{Dividend payment during the life of the option} \end{array}$$

If the Option is American Option, there is a risk of early assignment on the contracts you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	Price at position open ($)	Price at the first expiry ($)	Strike ($)
Near-expiry Call	18.50	23.75	832.50
Near-expiry Put	22.50	12.85	832.50
Far-expiry Call	23.90	24.45	832.50
Far-expiry Put	19.00	13.70	832.50
Underlying Equity	-	843.2500	-

Therefore, the payoff is

$$\begin{array}{rcll} Payoff_{T_1} & = & (P_{T_1}^{T_1} - P_0^{T_1} - C_{T_1}^{T_1} + C_0^{T_1} + C_{T_1}^{T_2} - C_0^{T_2} - P_{T_1}^{T_2} + P_0^{T_2})\times m - fee\\ & = & (12.85 - 22.50 - 23.75 + 18.50 + 24.45 - 23.90 - 13.70 + 19.00)\times100 - 1.00\times4\\ & = & -909.00\\ \end{array}$$

So, the strategy loses $909.

The following algorithm implements a long jelly roll Option strategy:

```C#
public class JellyRollStrategy : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().JellyRoll(5m, 30, 60));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested ||
            !slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            return;
        }

        // Select expiry dates and strike price
        var strike = chain.OrderBy(x => Math.Abs(chain.Underlying.Price - x.Strike)).First().Strike;
        var contracts = chain.Where(x => x.Strike == strike).ToList();
        var farExpiry = contracts.Max(x => x.Expiry);
        var nearExpiry = contracts.Min(x => x.Expiry);

        // Order Strategy
        var jellyRoll = OptionStrategies.JellyRoll(_symbol, strike, nearExpiry, farExpiry);
        Buy(jellyRoll, 1);
    }
}
```

Option Strategies

Short Jelly Roll

Introduction

A Short Jelly Roll , or simply Short Roll , is a combination of a short call calendar spread and a long put calendar spread . It is the inverse of a jelly roll . It consist of selling a put and buying a call of the same expiry, as well as buying a put and selling a call with a further expiry, where all of the contracts have the same strike prices. This strategy serves as an arbitrage on Option mispricing due to the temporary disparity between the call spread and the put spread synthetic portfolios. It is a delta -, gamma - , vega -, and theta -neutral strategy, but sensitive to rho (interest rate) and phi (dividend yield).

Implementation

Follow these steps to implement the short jelly roll strategy:

1. In the `Initialize` method, set the start date, set the end date, and create an Option universe .

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(100000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(x => x.IncludeWeeklys().JellyRoll(5m, 30, 60));
}
```

2. In the `OnData` method, select the expiry and strikes of the contracts in the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested) return;

    // Get the OptionChain
    if (!slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select expiry dates and strike price
    var strike = chain.OrderBy(x => Math.Abs(chain.Underlying.Price - x.Strike)).First().Strike;
    var contracts = chain.Where(x => x.Strike == strike).ToList();
    var farExpiry = contracts.Max(x => x.Expiry);
    var farExpiry = contracts.Max(x => x.Expiry);
    var nearerExpiries = contracts.Where(x => x.Expiry < farExpiry);
    if (!nearerExpiries.Any()) return;
    var nearExpiry = nearerExpiries.Min(x => x.Expiry);
```

3. In the `OnData` method, select the contracts and place the order.

   Approach A: Call the `OptionStrategies.ShortJellyRoll` method with the details of each leg and then pass the result to the `Buy` method.

```C#
var shortJellyRoll = OptionStrategies.ShortJellyRoll(_symbol, strike, nearExpiry, farExpiry);
Buy(shortJellyRoll, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

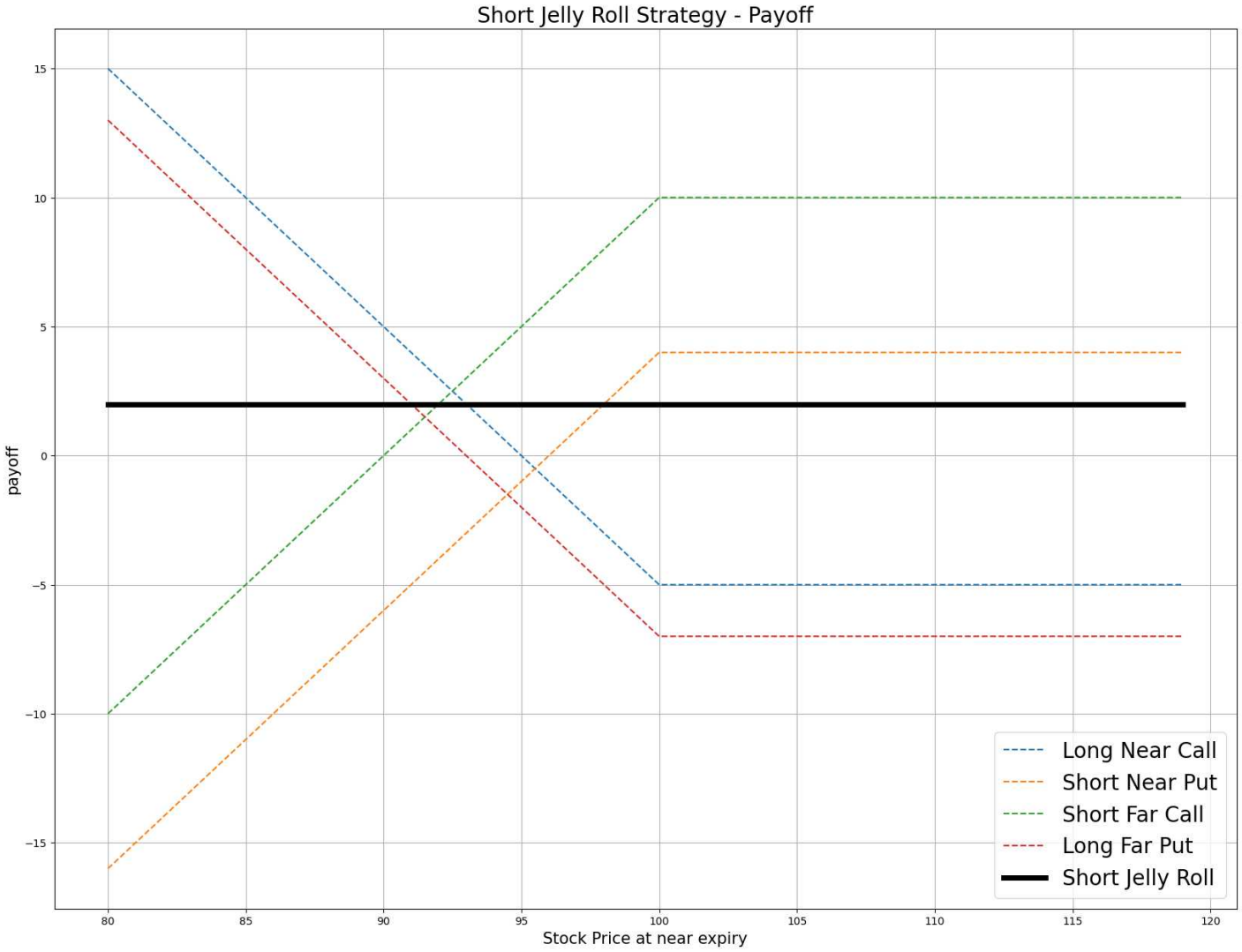
```C#
// Select the call and put contracts
var nearCall = contracts.Single(x => x.Expiry == nearExpiry && x.Right == OptionRight.Call);
var farCall = contracts.Single(x => x.Expiry == farExpiry && x.Right == OptionRight.Call);
var nearPut = contracts.Single(x => x.Expiry == nearExpiry && x.Right == OptionRight.Put);
var farPut = contracts.Single(x => x.Expiry == farExpiry && x.Right == OptionRight.Put);

var legs = new List<Leg>()
{
    Leg.Create(nearCall.Symbol, 1),
    Leg.Create(farCall.Symbol, -1),
    Leg.Create(nearPut.Symbol, -1),
    Leg.Create(farPut.Symbol, 1),
};
ComboMarketOrder(legs, 1);
```

Strategy Payoff

This is a delta-, gamma-, vega-, and theta-neutral strategy. The payoff is

$$\begin{array}{rcll} C_{T_1}^{T_1} & = & (S_{T_1} - K)^{+}\\ P_{T_1}^{T_1} & = & (K - S_{T_1})^{+}\\ Payoff_{T_1} & = & (C_{T_1}^{T_1} - C_{T_0}^{T_1} - P_{T_1}^{T_1} + P_{T_0}^{T_1} + P_{T_1}^{T_2} - P_{T_0}^{T_2} - C_{T_1}^{T_2} + C_{T_0}^{T_2})\times m - fee\\ \end{array} \begin{array}{rcll} \textrm{where} & C_{T_1}^{T_1} & = & \textrm{Market value of Call with expiry at T1 at time T1}\\ & C_{T_1}^{T_2} & = & \textrm{Market value of Call with expiry at T2 at time T1}\\ & P_{T_1}^{T_1} & = & \textrm{Market value of Put with expiry at T1 at time T1}\\ & P_{T_1}^{T_2} & = & \textrm{Market value of Put with expiry at T2 at time T1}\\ & S_{T_1} & = & \textrm{Underlying asset price at time T1}\\ & K & = & \textrm{Strike price}\\ & Payoff_{T_1} & = & \textrm{Payout total at time T1}\\ & C_{T_0}^{T_1} & = & \textrm{Market value of Call with expiry at T1 when the trade opened}\\ & C_{T_0}^{T_2} & = & \textrm{Market value of Call with expiry at T2 when the trade opened}\\ & P_{T_0}^{T_1} & = & \textrm{Market value of Put with expiry at T2 when the trade opened}\\ & P_{T_0}^{T_2} & = & \textrm{Market value of Put with expiry at T2 when the trade opened}\\ & m & = & \textrm{Contract multiplier}\\ & T_1 & = & \textrm{Time T1 as the near expiration}\\ & T_2 & = & \textrm{Time T2 as the far expiration} \end{array}$$

The following chart shows the payoff at expiration:



The payoff is dependent on the market prices of the options, but in theory, if assuming call-put parity exists, the expected payoff would be

$$\begin{array}{rcll} Payoff_{T_1} & = & Payoff_{\textrm{put calendar spread}} - Payoff_{\textrm{call calendar spread}}\\ & = & D - K \times (T_2 - T_1) \times r \end{array} \begin{array}{rcl} \textrm{where} & r & = & \textrm{Continuous compounding interest rate}\\ & D & = & \textrm{Dividend payment during the life of the option} \end{array}$$

If the Option is American Option, there is a risk of early assignment on the contracts you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	Price at position open ($)	Price at the first expiry ($)	Strike ($)
Near-expiry Call	22.80	23.75	832.50
Near-expiry Put	18.10	12.85	832.50
Far-expiry Call	19.50	24.45	832.50
Far-expiry Put	23.50	13.70	832.50
Underlying Equity	-	843.2500	-

Therefore, the payoff is

$$\begin{array}{rcll} Payoff_{T_1} & = & (C_{T_1}^{T_1} - C_0^{T_1} - P_{T_1}^{T_1} + P_0^{T_1} + P_{T_1}^{T_2} - P_0^{T_2} - C_{T_1}^{T_2} + C_0^{T_2})\times m - fee\\ & = & (23.75 - 22.80 - 12.85 + 18.10 + 13.70 - 23.50 - 24.45 + 19.50)\times100 - 1.00\times4\\ & = & -859.00\\ \end{array}$$

So, the strategy loses $859.

The following algorithm implements a short jelly roll Option strategy:

```C#
public class ShortJellyRollStrategy : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(x => x.IncludeWeeklys().JellyRoll(5m, 30, 60));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested ||
            !slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            return;
        }

        // Select expiry dates and strike price
        var strike = chain.OrderBy(x => Math.Abs(chain.Underlying.Price - x.Strike)).First().Strike;
        var contracts = chain.Where(x => x.Strike == strike).ToList();
        var farExpiry = contracts.Max(x => x.Expiry);
        var nearExpiry = contracts.Min(x => x.Expiry);

        // Order Strategy
        var shortJellyRoll = OptionStrategies.ShortJellyRoll(_symbol, strike, nearExpiry, farExpiry);
        Buy(shortJellyRoll, 1);
    }
}
```

Option Strategies

Bear Call Ladder

Introduction

Bear call ladder , also known as short call ladder , is a combination of a bear call spread and a long call with a higher strike price than the 2 legs of the call spread. All calls have the same underlying Equity and expiration date. This strategy profits from increasing volatility of the underlying asset. For instance, the underlying price moves away from its current price.

Implementation

Follow these steps to implement the bear call ladder strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(1000000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().CallLadder(30, 5, 0, -5));
}
```

2. In the `OnData` method, select the expiration and strikes of the contracts in the strategy legs.

```
C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select the call Option contracts with the furthest expiry
    var expiry = chain.Max(x => x.Expiry);
    var calls = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Call);
    if (calls.Count() == 0) return;

    // Select the strike prices from the remaining contracts
    var strikes = calls.Select(x => x.Strike).Distinct().OrderBy(x => x).ToList();
    if (strikes.Count < 3)
    {
        return;
    }

    var lowStrike = strikes[0];
    var middleStrike = strikes[1];
    var highStrike = strikes[2];
```

3. In the `OnData` method, select the contracts and place the orders.

   Approach A: Call the `OptionStrategies.BearCallLadder` method with the details of each leg and then pass the result to the `Buy` method.

```
var optionStrategy = OptionStrategies.BearCallLadder(_symbol, lowStrike, middleStrike, highStrike, expiry);
Buy(optionStrategy, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

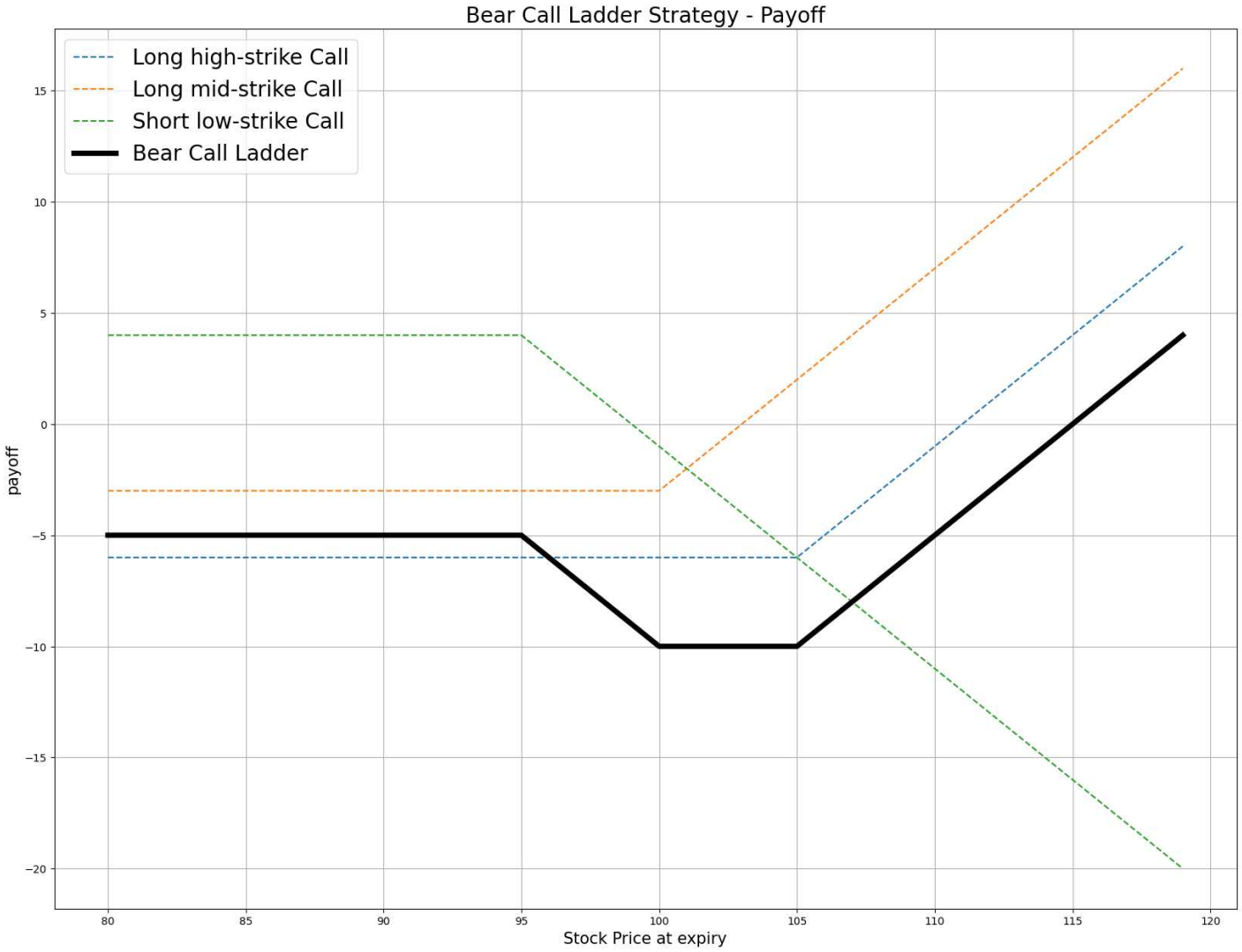
```
var lowStrikeCall = calls.Single(x => x.Strike == lowStrike);
var middleStrikeCall = calls.Single(x => x.Strike == middleStrike);
var highStrikeCall = calls.Single(x => x.Strike == highStrike);

var legs = new List<Leg>()
{
    Leg.Create(lowStrikeCall.Symbol, -1),
    Leg.Create(middleStrikeCall.Symbol, 1),
    Leg.Create(highStrikeCall.Symbol, 1)
};
ComboMarketOrder(legs, 1, true);
```

Strategy Payoff

The bear call ladding is an unlimited-profit-limited-risk strategy. The payoff is

$$\begin{array}{rcll} C^{low}_T & = & (S_T - K^{low})^{+}\\ C^{mid}_T & = & (S_T - K^{mid})^{+}\\ C^{high}_T & = & (S_T - K^{high})^{+}\\ Payoff_T & = & (C^{low}_0 - C^{low}_T + C^{mid}_T - C^{mid}_0 + C^{high}_T - C^{high}_0)\times m - fee\\ \end{array} \begin{array}{rcll} \textrm{where} & C^{low}_T & = & \textrm{Lower-strike call value at time T}\\ & C^{mid}_T & = & \textrm{Middle-strike call value at time T}\\ & C^{high}_T & = & \textrm{Higher-strike call value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^{low} & = & \textrm{Lower-strike call strike price}\\ & K^{mid} & = & \textrm{Middle-strike call strike price}\\ & K^{high} & = & \textrm{Higher-strike call strike price}\\ & C^{low}_0 & = & \textrm{Lower-strike call value at position opening (credit received)}\\ & C^{mid}_0 & = & \textrm{Middle-strikeTM call value at position opening (debit paid)}\\ & C^{high}_0 & = & \textrm{Higher-strike call value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is unlimited, which occurs when the underlying price increases indefinitely.

The maximum loss is K^{low} - K^{mid} + C^{low}_0 - C^{mid}_0 - C^{high}_0, which occurs when the underlying price is between the two higher strike prices.

If the Option is American Option, there is a risk of early assignment on the contract you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	**Price ($)**	**Strike ($)**
Lower-Strike call	13.80	822.50
Middle-strike call	15.10	825.00
Higher-strike call	13.10	827.50
Underlying Equity at expiration	843.25	-

Therefore, the payoff is

\begin{array}{rcll} C^{low}_T & = & (S_T - K^{low})^{+}\\ & = & (843.25-822.50)^{+}\\ & = & 20.75\\ C^{mid}_T & = & (S_T - K^{mid})^{+}\\ & = & (843.25-825.00)^{+}\\ & = & 18.25\\ C^{high}_T & = & (S_T - K^{high})^{+}\\ & = & (843.25-827.50)^{+}\\ & = & 15.75\\ Payoff_T & = & (C^{low}_0 - C^{low}_T + C^{mid}_T - C^{mid}_0 + C^{high}_T - C^{high}_0)\times m - fee\\ & = & (13.80 - 20.75 + 18.25 - 15.10 + 15.75 - 13.10)\times100-1.00\times3\\ & = & -118\\ \end{array}

So, the strategy loses $118.

The following algorithm implements a bear call ladder Option strategy:

```C#
public class BearCallLadderStrategy : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().CallLadder(30, 5, 0, -5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested ||
            !slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            return;
        }

        // Select the call Option contracts with the furthest expiry
        var expiry = chain.Max(x => x.Expiry);
        var calls = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Call);
        if (calls.Count() == 0) return;

        // Select the strike prices from the remaining contracts
        var strikes = calls.Select(x => x.Strike).Distinct().OrderBy(x => x).ToList();
        if (strikes.Count < 3)
        {
            return;
        }

        var lowStrike = strikes[0];
        var middleStrike = strikes[1];
        var highStrike = strikes[2];

        // Order Strategy
        var optionStrategy = OptionStrategies.BearCallLadder(_symbol, lowStrike, middleStrike, highStrike, expiry);
        Buy(optionStrategy, 1);
    }
}
```

Option Strategies

Bear Put Ladder

Introduction

Bear put ladder , also known as long put ladder , is a combination of a bear put spread and short put with a lower strike price than the 2 legs of the put spread. All puts have the same underlying Equity and expiration date. This strategy profits from low volatility of the underlying asset. For instance, the underlying price stays similar to its current price.

Implementation

Follow these steps to implement the bear put ladder strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(1000000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().PutLadder(30, 5, 0, -5));
}
```

2. In the `OnData` method, select the expiration and strikes of the contracts in the strategy legs.

```
C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select the put Option contracts with the furthest expiry
    var expiry = chain.Max(x => x.Expiry);
    var puts = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Put);
    if (puts.Count() == 0) return;

    // Select the strike prices from the remaining contracts
    var strikes = puts.Select(x => x.Strike).Distinct().OrderBy(x => x).ToList();
    if (strikes.Count < 3)
    {
        return;
    }

    var lowStrike = strikes[0];
    var middleStrike = strikes[1];
    var highStrike = strikes[2];
```

3. In the `OnData` method, select the contracts and place the orders.

    Approach A: Put the `OptionStrategies.BearPutLadder` method with the details of each leg and then pass the result to the `Buy` method.

```C#
var optionStrategy = OptionStrategies.BearPutLadder(_symbol, highStrike, middleStrike, lowStrike, expiry);
Buy(optionStrategy, 1);
```

Approach B: Create a list of `Leg` objects and then put the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

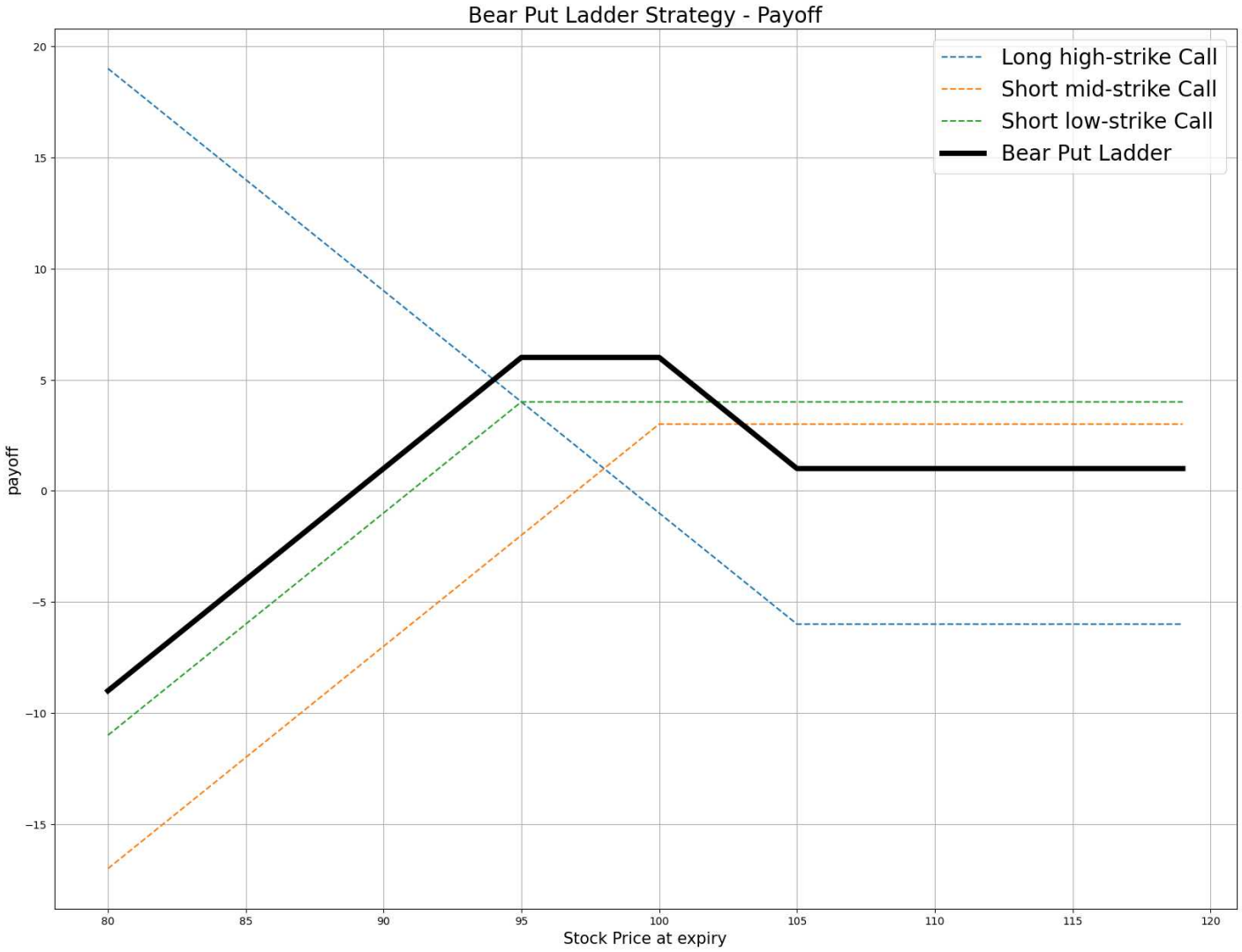
```C#
var lowStrikePut = puts.Single(x => x.Strike == lowStrike);
var middleStrikePut = puts.Single(x => x.Strike == middleStrike);
var highStrikePut = puts.Single(x => x.Strike == highStrike);

var legs = new List<Leg>()
{
    Leg.Create(lowStrikePut.Symbol, -1),
    Leg.Create(middleStrikePut.Symbol, -1),
    Leg.Create(highStrikePut.Symbol, 1)
};
ComboMarketOrder(legs, 1, true);
```

Strategy Payoff

The bear put ladding is an limited-profit strategy. The payoff is

$$\begin{array}{rcll} P^{low}_T & = & (K^{low} - S_T)^{+}\\ P^{mid}_T & = & (K^{mid} - S_T)^{+}\\ P^{high}_T & = & (K^{high} - S_T)^{+}\\ Payoff_T & = & (P^{low}_0 - P^{low}_T + P^{mid}_0 - P^{mid}_T + P^{high}_T - P^{high}_0)\times m - fee\\ \end{array} \begin{array}{rcl} \textrm{where} & P^{low}_T & = & \textrm{Lower-strike put value at time T}\\ & P^{mid}_T & = & \textrm{Middle-strike put value at time T}\\ & P^{high}_T & = & \textrm{Higher-strike put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^{low} & = & \textrm{Lower-strike put strike price}\\ & K^{mid} & = & \textrm{Middle-strike put strike price}\\ & K^{high} & = & \textrm{Higher-strike put strike price}\\ & P^{low}_0 & = & \textrm{Lower-strike put value at position opening (credit received)}\\ & P^{mid}_0 & = & \textrm{Middle-strikeTM put value at position opening (debit paid)}\\ & P^{high}_0 & = & \textrm{Higher-strike put value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is $K^{high} - K^{mid} + P^{low}_0 + P^{mid}_0 - P^{high}_0$, which occurs when the underlying price is between the two lower strike prices.

The maximum loss is $K^{high} - K^{mid} - K^{low} + P^{low}_0 + P^{mid}_0 - P^{high}_0$, which occurs when the underlying price decreases to $0.

If the Option is American Option, there is a risk of early assignment on the contract you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	Price ($)	Strike ($)
Lower-Strike put	3.80	822.50
Middle-strike put	4.70	825.00
Higher-strike put	7.80	827.50
Underlying Equity at expiration	843.25	-

Therefore, the payoff is

$$\begin{array}{rcll} P^{low}_T &=& (K^{low} - S_T)^{+}\\ &=& (822.50-843.25)^{+}\\ &=& 0\\ P^{mid}_T &=& (K^{mid} - S_T)^{+}\\ &=& (825.00-843.25)^{+}\\ &=& 0\\ P^{high}_T &=& (K^{high} - S_T)^{+}\\ &=& (827.50-843.25)^{+}\\ &=& 0\\ Payoff_T &=& (P^{low}_0 - P^{low}_T + P^{mid}_0 - P^{mid}_T + P^{high}_T - P^{high}_0)\times m - fee\\ &=& (3.80 - 0 + 4.70$$

- 0 + 0 - 7.80)\times100-1.00\times3\\ & = & 67\\ \end{array}

So, the strategy gains $67.

The following algorithm implements a bear put ladder Option strategy:

```C#
public class BearPutLadderStrategy : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().PutLadder(30, 5, 0, -5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested ||
            !slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            return;
        }

        // Select the call Option contracts with the furthest expiry
        var expiry = chain.Max(x => x.Expiry);
        var puts = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Put);
        if (puts.Count() == 0) return;

        // Select the strike prices from the remaining contracts
        var strikes = puts.Select(x => x.Strike).Distinct().OrderBy(x => x).ToList();
        if (strikes.Count < 3)
        {
            return;
        }

        var lowStrike = strikes[0];
        var middleStrike = strikes[1];
        var highStrike = strikes[2];

        // Order Strategy
        var optionStrategy = OptionStrategies.BearPutLadder(_symbol, highStrike, middleStrike, lowStrike, expiry);
        Buy(optionStrategy, 1);
    }
}
```

Option Strategies

Bull Call Ladder

Introduction

Bull call ladder , also known as long call ladder , is a combination of a bull call spread and a short call with a higher strike price than the 2 legs of the call spread. All calls have the same underlying Equity and expiration date. This strategy profits from low volatility of the underlying asset. For instance, the underlying price stays similar to its current price.

Implementation

Follow these steps to implement the bull call ladder strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(1000000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().CallLadder(30, 5, 0, -5));
}
```

2. In the `OnData` method, select the expiration and strikes of the contracts in the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select the call Option contracts with the furthest expiry
    var expiry = chain.Max(x => x.Expiry);
    var calls = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Call);
    if (calls.Count() == 0) return;

    // Select the strike prices from the remaining contracts
    var strikes = calls.Select(x => x.Strike).Distinct().OrderBy(x => x).ToList();
    if (strikes.Count < 3)
    {
        return;
    }

    var lowStrike = strikes[0];
    var middleStrike = strikes[1];
    var highStrike = strikes[2];
```

3. In the `OnData` method, select the contracts and place the orders.

    Approach A: Call the `OptionStrategies.BullCallLadder` method with the details of each leg and then pass the result to the `Buy` method.

```
var optionStrategy = OptionStrategies.BullCallLadder(_symbol, lowStrike, middleStrike, highStrike, expiry);
Buy(optionStrategy, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

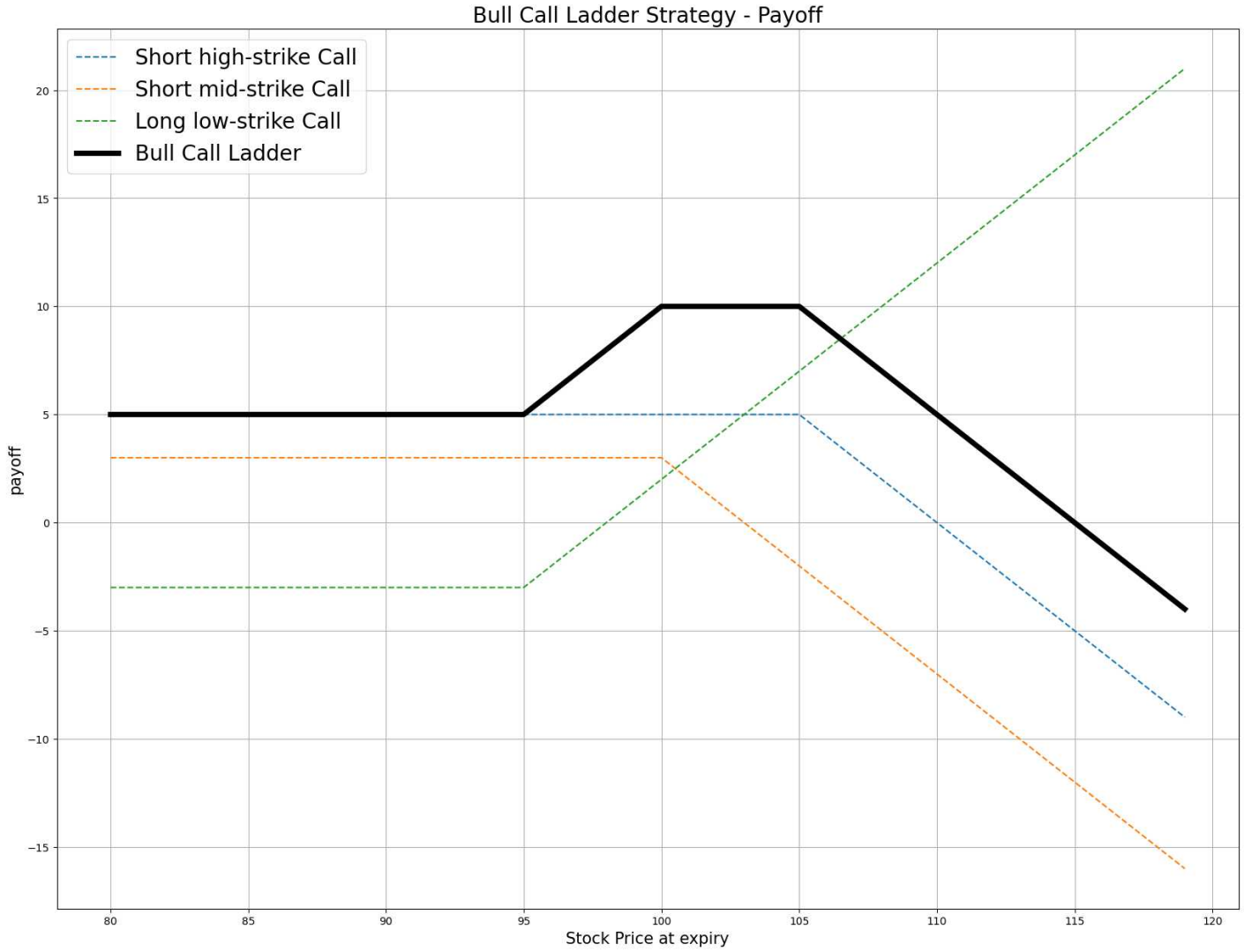
```
var lowStrikeCall = calls.Single(x => x.Strike == lowStrike);
var middleStrikeCall = calls.Single(x => x.Strike == middleStrike);
var highStrikeCall = calls.Single(x => x.Strike == highStrike);

var legs = new List<Leg>()
{
    Leg.Create(lowStrikeCall.Symbol, 1),
    Leg.Create(middleStrikeCall.Symbol, -1),
    Leg.Create(highStrikeCall.Symbol, -1)
};
ComboMarketOrder(legs, 1, true);
```

Strategy Payoff

The bull call spread is an limited-profit strategy. The payoff is

$$\begin{array}{rcll} C^{low}_T & = & (S_T - K^{low})^{+}\\ C^{mid}_T & = & (S_T - K^{mid})^{+}\\ C^{high}_T & = & (S_T - K^{high})^{+}\\ Payoff_T & = & (C^{low}_T - C^{low}_0 + C^{mid}_0 - C^{mid}_T + C^{high}_0 - C^{high}_T)\times m - fee\\ \end{array} \begin{array}{rcl} \textrm{where} & C^{low}_T & = & \textrm{Lower-strike call value at time T}\\ & C^{mid}_T & = & \textrm{Middle-strike call value at time T}\\ & C^{high}_T & = & \textrm{Higher-strike call value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^{low} & = & \textrm{Lower-strike call strike price}\\ & K^{mid} & = & \textrm{Middle-strike call strike price}\\ & K^{high} & = & \textrm{Higher-strike call strike price}\\ & C^{low}_0 & = & \textrm{Lower-strike call value at position opening (credit received)}\\ & C^{mid}_0 & = & \textrm{Middle-strikeTM call value at position opening (debit paid)}\\ & C^{high}_0 & = & \textrm{Higher-strike call value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is K^{mid} - K^{low} - C^{low}_0 + C^{mid}_0 + C^{high}_0, which occurs when the underlying price is between the two higher strike prices.

The maximum loss is unlimited, which occurs when the underlying price increases indefinitely.

If the Option is American Option, there is a risk of early assignment on the contract you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	**Price ($)**	**Strike ($)**
Lower-Strike call	17.10	822.50
Middle-strike call	12.00	825.00
Higher-strike call	10.90	827.50
Underlying Equity at expiration	843.25	-

Therefore, the payoff is

\begin{array}{rcll} C^{low}_T & = & (S_T - K^{low})^{+}\\ & = & (843.25-822.50)^{+}\\ & = & 20.75\\ C^{mid}_T & = & (S_T - K^{mid})^{+}\\ & = & (843.25-825.00)^{+}\\ & = & 18.25\\ C^{high}_T & = & (S_T - K^{high})^{+}\\ & = & (843.25-827.50)^{+}\\ & = & 15.75\\ Payoff_T & = & (C^{low}_T - C^{low}_0 + C^{mid}_0 - C^{mid}_T + C^{high}_0 - C^{high}_T)\times m - fee\\ & = & (20.75 - 17.10 + 12.00 - 18.25 + 10.90 - 15.75)\times100-1.00\times3\\ & = & -748\\ \end{array}

So, the strategy loses $748.

The following algorithm implements a bull call ladder Option strategy:

```C#
public class BullCallLadderStrategy : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().CallLadder(30, 5, 0, -5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested ||
            !slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            return;
        }

        // Select the call Option contracts with the furthest expiry
        var expiry = chain.Max(x => x.Expiry);
        var calls = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Call);
        if (calls.Count() == 0) return;

        // Select the strike prices from the remaining contracts
        var strikes = calls.Select(x => x.Strike).Distinct().OrderBy(x => x).ToList();
        if (strikes.Count < 3)
        {
            return;
        }

        var lowStrike = strikes[0];
        var middleStrike = strikes[1];
        var highStrike = strikes[2];

        // Order Strategy
        var optionStrategy = OptionStrategies.BullCallLadder(_symbol, lowStrike, middleStrike, highStrike, expiry);
        Buy(optionStrategy, 1);
    }
}
```

Option Strategies

Bull Put Ladder

Introduction

Bull put ladder , also known as short put ladder , is a combination of a bull put spread and short put with a lower strike price than the 2 legs of the put spread. All puts have the same underlying Equity and expiration date. This strategy profits from increasing volatility of the underlying asset. For instance, the underlying price moves away from its current price.

Implementation

Follow these steps to implement the bull put ladder strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe .

```
C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(1000000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().PutLadder(30, 5, 0, -5));
}
```

2. In the `OnData` method, select the expiration and strikes of the contracts in the strategy legs.

```
C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select the put Option contracts with the furthest expiry
    var expiry = chain.Max(x => x.Expiry);
    var puts = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Put);
    if (puts.Count() == 0) return;

    // Select the strike prices from the remaining contracts
    var strikes = puts.Select(x => x.Strike).Distinct().OrderBy(x => x).ToList();
    if (strikes.Count < 3)
    {
        return;
    }

    var lowStrike = strikes[0];
    var middleStrike = strikes[1];
    var highStrike = strikes[2];
```

3. In the `OnData` method, select the contracts and place the orders.

   Approach A: Put the `OptionStrategies.BullPutLadder` method with the details of each leg and then pass the result to the `Buy` method.

```C#
var optionStrategy = OptionStrategies.BullPutLadder(_symbol, highStrike, middleStrike, lowStrike, expiry);
Buy(optionStrategy, 1);
```

Approach B: Create a list of `Leg` objects and then put the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

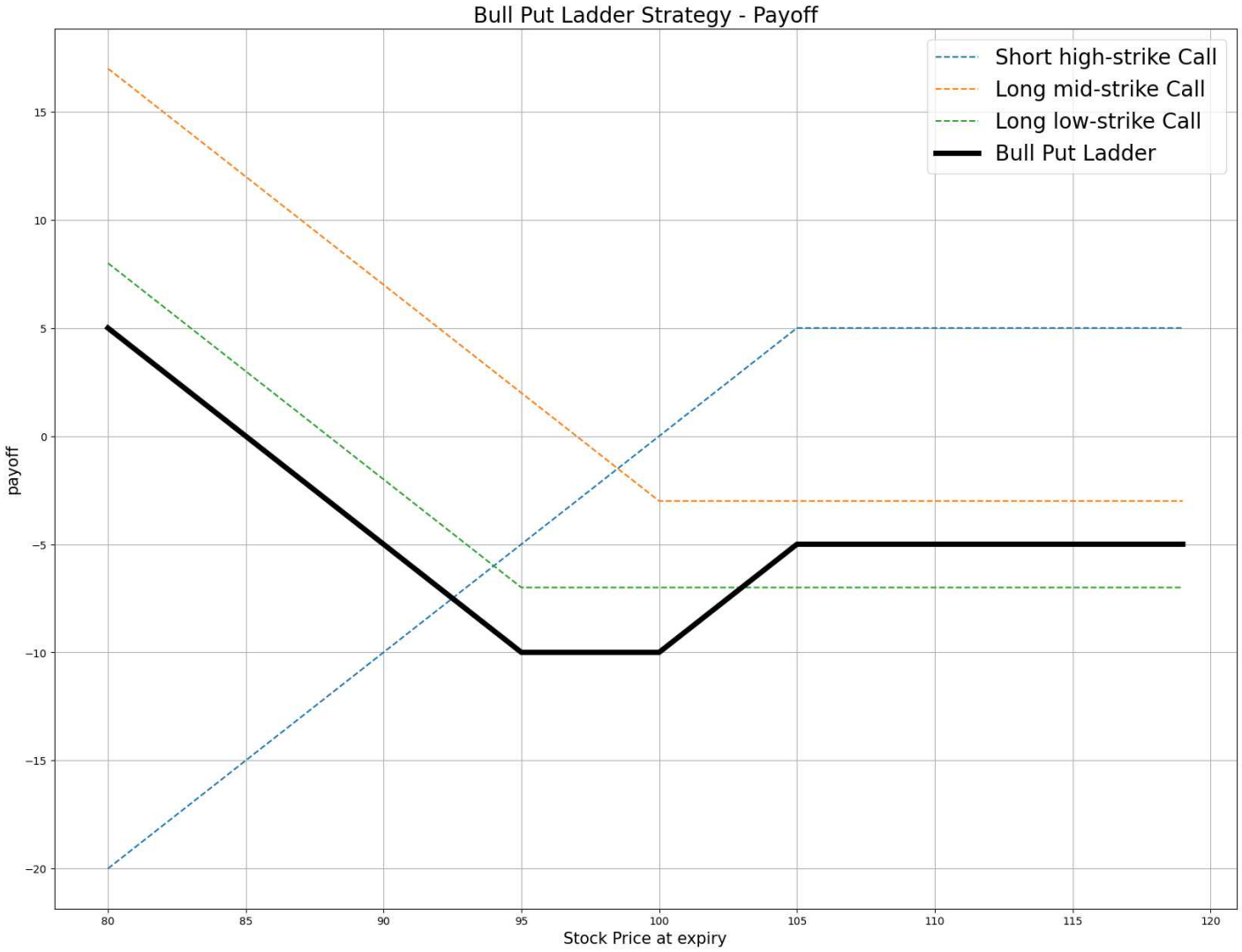
```C#
var lowStrikePut = puts.Single(x => x.Strike == lowStrike);
var middleStrikePut = puts.Single(x => x.Strike == middleStrike);
var highStrikePut = puts.Single(x => x.Strike == highStrike);

var legs = new List<Leg>()
{
    Leg.Create(lowStrikePut.Symbol, 1),
    Leg.Create(middleStrikePut.Symbol, 1),
    Leg.Create(highStrikePut.Symbol, -1)
};
ComboMarketOrder(legs, 1, true);
```

Strategy Payoff

The bull put spread is an limited-risk strategy. The payoff is

$$\begin{array}{rcll} P^{low}_T & = & (K^{low} - S_T)^{+}\\ P^{mid}_T & = & (K^{mid} - S_T)^{+}\\ P^{high}_T & = & (K^{high} - S_T)^{+}\\ Payoff_T & = & (P^{low}_T - P^{low}_0 + P^{mid}_T - P^{mid}_0 + P^{high}_0 - P^{high}_T)\times m - fee\\ \end{array} \begin{array}{rcl} \textrm{where} & P^{low}_T & = & \textrm{Lower-strike put value at time T}\\ & P^{mid}_T & = & \textrm{Middle-strike put value at time T}\\ & P^{high}_T & = & \textrm{Higher-strike put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^{low} & = & \textrm{Lower-strike put strike price}\\ & K^{mid} & = & \textrm{Middle-strike put strike price}\\ & K^{high} & = & \textrm{Higher-strike put strike price}\\ & P^{low}_0 & = & \textrm{Lower-strike put value at position opening (credit received)}\\ & P^{mid}_0 & = & \textrm{Middle-strikeTM put value at position opening (debit paid)}\\ & P^{high}_0 & = & \textrm{Higher-strike put value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is K^{mid} + K^{low} - K^{high} - P^{low}_0 - P^{mid}_0 + P^{high}_0, which occurs when the underlying price decreases to $0.

The maximum loss is K^{mid} - K^{high} - P^{low}_0 - P^{mid}_0 + P^{high}_0, which occurs when the underlying price is between the two lower strike prices.

If the Option is American Option, there is a risk of early assignment on the contract you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	Price ($)	Strike ($)
Lower-Strike put	6.00	822.50
Middle-strike put	4.70	825.00
Higher-strike put	5.60	827.50
Underlying Equity at expiration	843.25	-

Therefore, the payoff is

\begin{array}{rcll} P^{low}_T & = & (K^{low} - S_T)^{+}\\ & = & (822.50-843.25)^{+}\\ & = & 0\\ P^{mid}_T & = & (K^{mid} - S_T)^{+}\\ & = & (825.00-843.25)^{+}\\ & = & 0\\ P^{high}_T & = & (K^{high} - S_T)^{+}\\ & = & (827.50-843.25)^{+}\\ & = & 0\\ Payoff_T & = & (P^{low}_T - P^{low}_0 + P^{mid}_T - P^{mid}_0 + P^{high}_0 - P^{high}_T)\times m - fee\\ & = & (0 - 6.00 + 0 -

4.70 + 5.60 - 0)\times100-1.00\times3\\ & = & -513\\ \end{array}

So, the strategy loses $513.

The following algorithm implements a bull put ladder Option strategy:

```C#
public class BullPutLadderStrategy : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().PutLadder(30, 5, 0, -5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested ||
            !slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            return;
        }

        // Select the call Option contracts with the furthest expiry
        var expiry = chain.Max(x => x.Expiry);
        var puts = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Put);
        if (puts.Count() == 0) return;

        // Select the strike prices from the remaining contracts
        var strikes = puts.Select(x => x.Strike).Distinct().OrderBy(x => x).ToList();
        if (strikes.Count < 3)
        {
            return;
        }

        var lowStrike = strikes[0];
        var middleStrike = strikes[1];
        var highStrike = strikes[2];

        // Order Strategy
        var optionStrategy = OptionStrategies.BullPutLadder(_symbol, highStrike, middleStrike, lowStrike, expiry);
        Buy(optionStrategy, 1);
    }
}
```

Option Strategies

Long Call Backspread

Introduction

Long Call Backspread , consists of short 1 lower-strike call and long 2 higher strike calls. It is a combination of a bear call spread and a long call with the same strike price as the higher-strike leg the call spread. All calls have the same underlying Equity and expiration date. This strategy profits from increasing volatility of the underlying asset. For instance, the underlying price moves away from its current price.

Implementation

Follow these steps to implement the long call backspread strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe . You can use the `CallSpread` helper method in option universe filtering, since a call backspread consists of the same contracts as a call spread.

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(1000000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().CallSpread(20, 5));
}
```

2. In the `OnData` method, select the expiration and strikes of the contracts in the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select the call Option contracts with the furthest expiry
    var expiry = chain.Max(x => x.Expiry);
    var calls = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Call);
    if (calls.Count() == 0) return;

    // Select the strike prices from the remaining contracts
    var strikes = calls.Select(x => x.Strike).Distinct().OrderBy(x => x).ToList();
    if (strikes.Count < 2)
    {
        return;
    }

    var lowStrike = strikes[0];
    var highStrike = strikes[1];
```

3. In the `OnData` method, select the contracts and place the orders.

Approach A: Call the `OptionStrategies.CallBackspread` method with the details of each leg and then pass the result to the `Buy` method.

```C#
var optionStrategy = OptionStrategies.CallBackspread(_symbol, lowStrike, highStrike, expiry);
Buy(optionStrategy, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

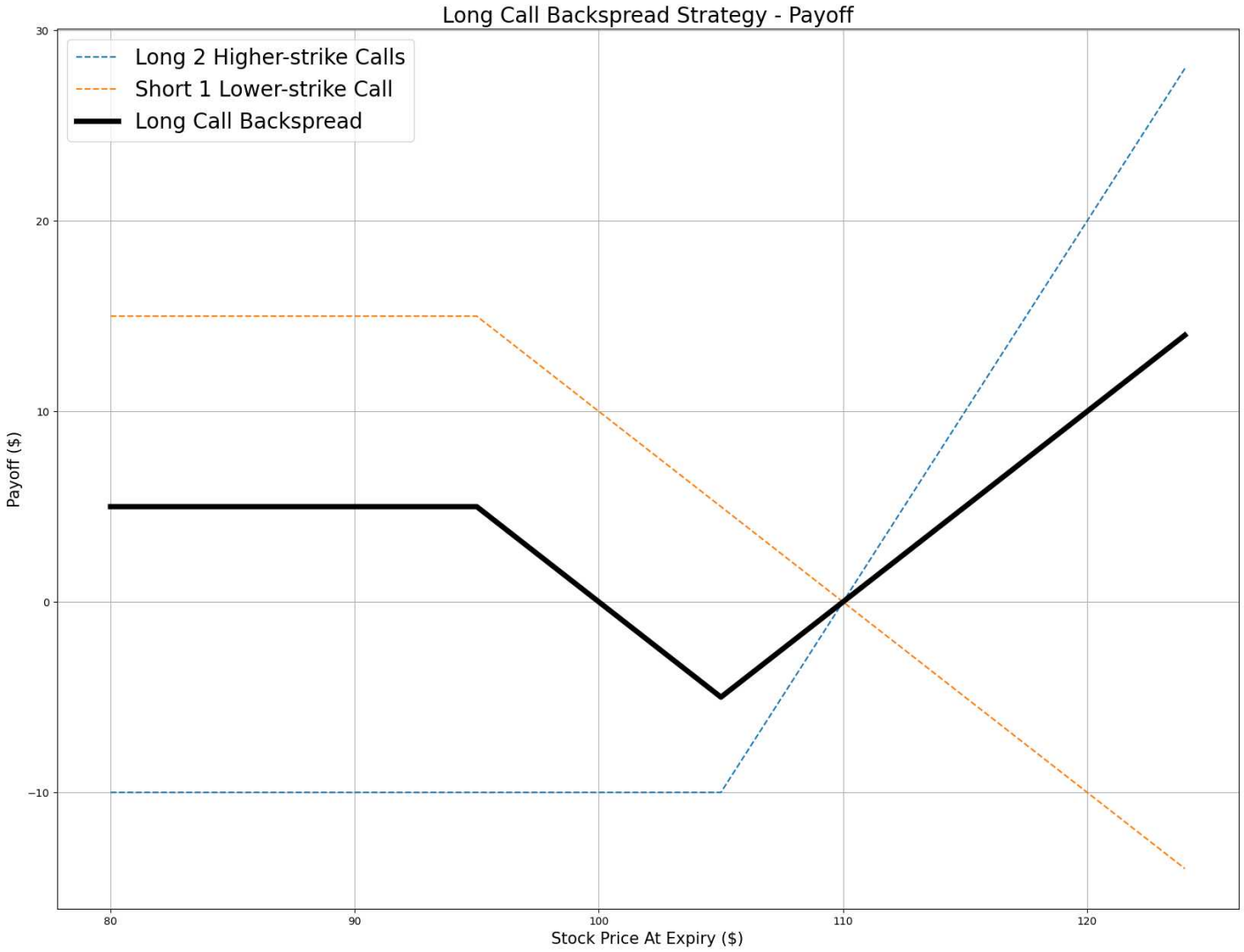
```C#
var lowStrikeCall = calls.Single(x => x.Strike == lowStrike);
var highStrikeCall = calls.Single(x => x.Strike == highStrike);

var legs = new List<Leg>()
{
    Leg.Create(lowStrikeCall.Symbol, -1),
    Leg.Create(highStrikeCall.Symbol, 2)
};
ComboMarketOrder(legs, 1, true);
```

Strategy Payoff

The long call backspread is an unlimited-profit-limited-risk strategy. The payoff is

$$\begin{array}{rcll} C^{low}_T & = & (S_T - K^{low})^{+}\\ C^{high}_T & = & (S_T - K^{high})^{+}\\ Payoff_T & = & (C^{low}_0 - C^{low}_T + C^{high}_T \times 2 - C^{high}_0 \times 2)\times m - fee\\ \end{array} \begin{array}{rcl} \textrm{where} & & \\ C^{low}_T & = & \textrm{Lower-strike call value at time T}\\ C^{high}_T & = & \textrm{Higher-strike call value at time T}\\ S_T & = & \textrm{Underlying asset price at time T}\\ K^{low} & = & \textrm{Lower-strike call strike price}\\ K^{high} & = & \textrm{Higher-strike call strike price}\\ C^{low}_0 & = & \textrm{Lower-strike call value at position opening (credit received)}\\ C^{high}_0 & = & \textrm{Higher-strike call value at position opening (debit paid)}\\ m & = & \textrm{Contract multiplier}\\ T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is unlimited, which occurs when the underlying price increases indefinitely.

The maximum loss is K^{low} - K^{high} + C^{low}_0 - C^{high}_0 \times 2, which occurs when the underlying price is exactly at the higher strike at expiry.

If the Option is American Option, there is a risk of early assignment on the contract you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	**Price ($)**	**Strike ($)**
Lower-Strike call	12.00	825.00
Higher-strike call	8.80	835.00
Underlying Equity at expiration	843.19	-

Therefore, the payoff is

\begin{array}{rcll} C^{low}_T & = & (S_T - K^{low})^{+}\\ & = & (843.19-825.00)^{+}\\ & = & 18.19\\ C^{high}_T & = & (S_T - K^{high})^{+}\\ & = & (843.19-835.00)^{+}\\ & = & 8.19\\ Payoff_T & = & (C^{low}_0 - C^{low}_T + C^{high}_T \times 2 - C^{high}_0 \times 2)\times m - fee\\ & = & (12.00 - 18.19 + 8.19\times2 - 8.80\times2)\times100-2.30\\ & = & -743.30\\ \end{array}

So, the strategy loses $743.30.

The following algorithm implements a long call backspread Option strategy:

```C#
public class BackspreadOptionStrategyAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);

        UniverseSettings.Asynchronous = true;
        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().CallSpread(20, 5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested ||
            !slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            return;
        }

        // Select the call Option contracts with the furthest expiry
        var expiry = chain.Max(x => x.Expiry);
        var calls = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Call);
        if (calls.Count() == 0) return;

        // Select the strike prices from the remaining contracts
        var strikes = calls.Select(x => x.Strike).Distinct().OrderBy(x => x).ToList();
        if (strikes.Count < 2)
        {
            return;
        }

        var lowStrike = strikes[0];
        var highStrike = strikes[1];

        var optionStrategy = OptionStrategies.CallBackspread(_symbol, lowStrike, highStrike, expiry);
        Buy(optionStrategy, 1);
    }
}
```

Option Strategies

Short Call Backspread

Introduction

Short Call Backspread , consists of long 1 lower-strike call and short 2 higher strike calls. It is a combination of a bull call spread and a short call with the same strike price as the higher-strike leg the call spread. All calls have the same underlying Equity and expiration date. This strategy profits from stable, consistent price of the underlying asset. For instance, the underlying price stays at its current price.

Implementation

Follow these steps to implement the short call backspread strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe . You can use the `CallSpread` helper method in option universe filtering, since a call backspread consists of the same contracts as a call spread.

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(1000000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().CallSpread(20, 5));
}
```

2. In the `OnData` method, select the expiration and strikes of the contracts in the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select the call Option contracts with the furthest expiry
    var expiry = chain.Max(x => x.Expiry);
    var calls = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Call);
    if (calls.Count() == 0) return;

    // Select the strike prices from the remaining contracts
    var strikes = calls.Select(x => x.Strike).Distinct().OrderBy(x => x).ToList();
    if (strikes.Count < 2)
    {
        return;
    }

    var lowStrike = strikes[0];
    var highStrike = strikes[1];
```

3. In the `OnData` method, select the contracts and place the orders.

Approach A: Call the `OptionStrategies.ShortCallBackspread` method with the details of each leg and then pass the result to the `Buy` method.

```
C#
var optionStrategy = OptionStrategies.ShortCallBackspread(_symbol, lowStrike, highStrike, expiry);
Buy(optionStrategy, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

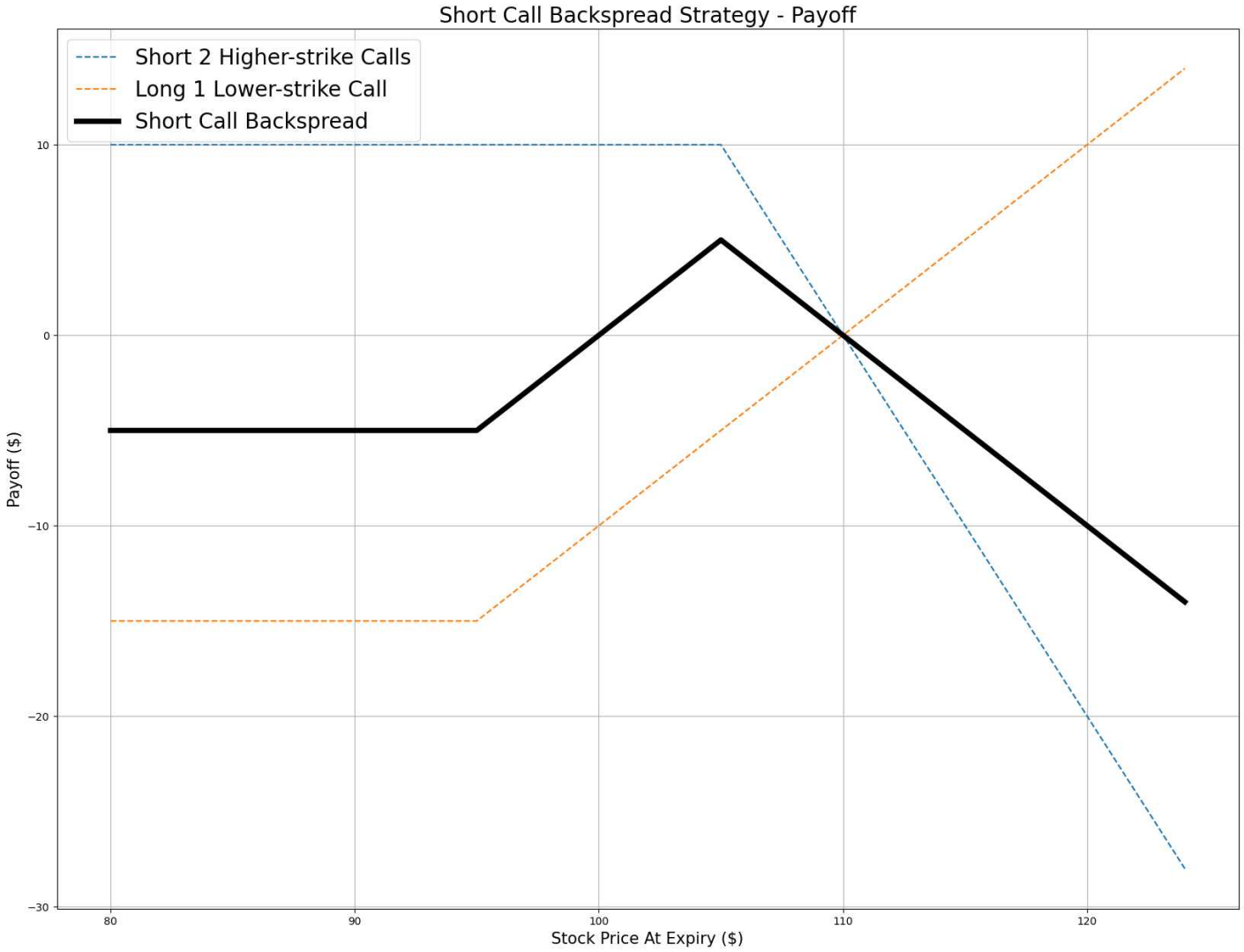
```
C#
var lowStrikeCall = calls.Single(x => x.Strike == lowStrike);
var highStrikeCall = calls.Single(x => x.Strike == highStrike);

var legs = new List<Leg>()
{
    Leg.Create(lowStrikeCall.Symbol, 1),
    Leg.Create(highStrikeCall.Symbol, -2)
};
ComboMarketOrder(legs, 1, true);
```

Strategy Payoff

The short call backspread is an limited-profit-unlimited-risk strategy. The payoff is

$$\begin{array}{rcll} C^{low}_T & = & (S_T - K^{low})^{+}\\ C^{high}_T & = & (S_T - K^{high})^{+}\\ Payoff_T & = & (C^{low}_T - C^{low}_0 + C^{high}_0 \times 2 - C^{high}_T \times 2)\times m - fee\\ \end{array} \begin{array}{rcll} \textrm{where} & & & \\ C^{low}_T & = & \textrm{Lower-strike call value at time T}\\ C^{high}_T & = & \textrm{Higher-strike call value at time T}\\ S_T & = & \textrm{Underlying asset price at time T}\\ K^{low} & = & \textrm{Lower-strike call strike price}\\ K^{high} & = & \textrm{Higher-strike call strike price}\\ C^{low}_0 & = & \textrm{Lower-strike call value at position opening (credit received)}\\ C^{high}_0 & = & \textrm{Higher-strike call value at position opening (debit paid)}\\ m & = & \textrm{Contract multiplier}\\ T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is $K^{high} - K^{low} - C^{low}_0 + C^{high}_0 \times 2$, which occurs when the underlying price is exactly at the higher strike at expiry.

The maximum loss is unlimited, which occurs when the underlying price increases indefinitely.

If the Option is American Option, there is a risk of early assignment on the contract you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	**Price ($)**	**Strike ($)**
Lower-Strike call	15.10	825.00
Higher-strike call	8.00	835.00
Underlying Equity at expiration	843.19	-

Therefore, the payoff is

$$\begin{array}{rcll} C^{low}_T & = & (S_T - K^{low})^{+}\\ & = & (843.19-825.00)^{+}\\ & = & 18.19\\ C^{high}_T & = & (S_T - K^{high})^{+}\\ & = & (843.19-835.00)^{+}\\ & = & 8.19\\ Payoff_T & = & (C^{low}_T - C^{low}_0 - C^{high}_T \times 2 + C^{high}_0 \times 2)\times m - fee\\ & = & (18.19 - 15.10 - 8.19\times2 + 8.00\times2)\times100-2.30\\ & = & 268.70\\ \end{array}$$

So, the strategy gains $268.70.

The following algorithm implements a short call backspread Option strategy:

```C#
public class BackspreadOptionStrategyAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);

        UniverseSettings.Asynchronous = true;
        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().CallSpread(20, 5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested ||
            !slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            return;
        }

        // Select the call Option contracts with the furthest expiry
        var expiry = chain.Max(x => x.Expiry);
        var calls = chain.Where(x => x.Expiry == expiry && x.Right == OptionRight.Call);
        if (calls.Count() == 0) return;

        // Select the strike prices from the remaining contracts
        var strikes = calls.Select(x => x.Strike).Distinct().OrderBy(x => x).ToList();
        if (strikes.Count < 2)
        {
            return;
        }

        var lowStrike = strikes[0];
        var highStrike = strikes[1];

        var optionStrategy = OptionStrategies.ShortCallBackspread(_symbol, lowStrike, highStrike, expiry);
        Buy(optionStrategy, 1);
    }
}
```

Option Strategies

Long Put Backspread

Introduction

Long Put Backspread , consists of short 1 higher-strike put and long 2 lower-strike puts. It is a combination of a bull put spread and a long put with the same strike price as the lower-strike leg the put spread. All puts have the same underlying Equity and expiration date. This strategy profits from increasing volatility of the underlying asset. For instance, the underlying price moves away from its current price.

Implementation

Follow these steps to implement the long put backspread strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe . You can use the `PutSpread` helper method in option universe filtering, since a put backspread consists of the same contracts as a put spread.

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(1000000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().PutSpread(20, 5));
}
```

2. In the `OnData` method, select the expiration and strikes of the contracts in the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select the put Option contracts with the furthest expiry
    var expiry = chain.Max(x => x.Expiry);
    var puts = chain.Where(x => x.Expiry == expiry);
    if (puts.Count() == 0) return;

    // Select the strike prices from the remaining contracts
    var strikes = puts.Select(x => x.Strike).Distinct().OrderBy(x => x).ToList();
    if (strikes.Count < 2)
    {
        return;
    }

    var lowStrike = strikes[0];
    var highStrike = strikes[1];
```

3. In the `OnData` method, select the contracts and place the orders.

Approach A: Call the `OptionStrategies.PutBackspread` method with the details of each leg and then pass the result to the `Buy` method.

```
C#
var optionStrategy = OptionStrategies.PutBackspread(_symbol, highStrike, lowStrike, expiry);
Buy(optionStrategy, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

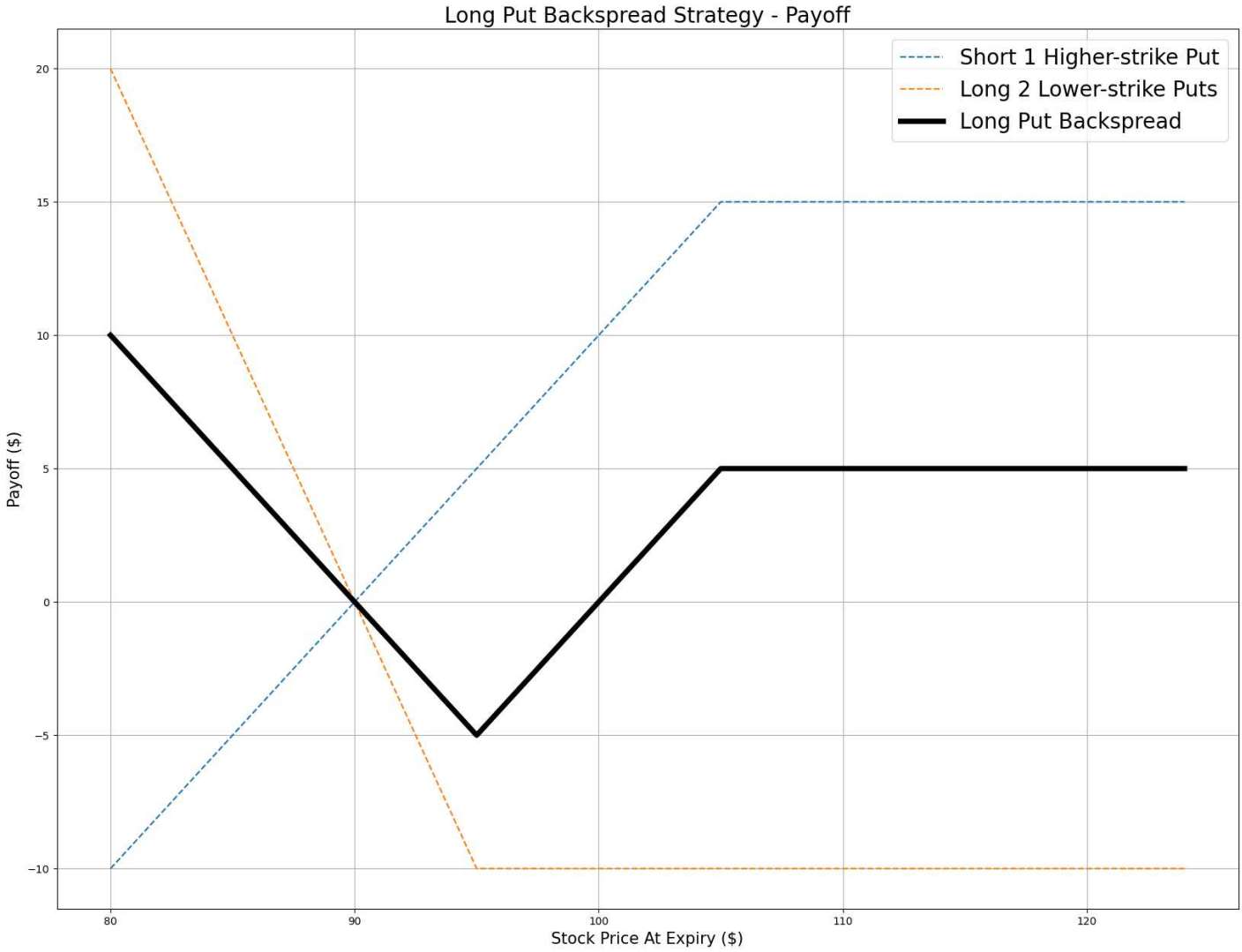
```
C#
var lowStrikePut = puts.Single(x => x.Strike == lowStrike);
var highStrikePut = puts.Single(x => x.Strike == highStrike);

var legs = new List<Leg>()
{
    Leg.Create(lowStrikePut.Symbol, 2),
    Leg.Create(highStrikePut.Symbol, -1)
};
ComboMarketOrder(legs, 1, true);
```

Strategy Payoff

The long put backspread is an unlimited-profit-limited-risk strategy. The payoff is

$$\begin{array}{rcll} P^{low}_T & = & (K^{low} - S_T)^{+}\\ P^{high}_T & = & (K^{high} - S_T)^{+}\\ Payoff_T & = & (P^{high}_0 - P^{high}_T + P^{low}_T \times 2 - P^{low}_0 \times 2)\times m - fee\\ \end{array} \begin{array}{rcll} \textrm{where} & P^{low}_T & = & \textrm{Lower-strike put value at time T}\\ & P^{high}_T & = & \textrm{Higher-strike put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^{low} & = & \textrm{Lower-strike put strike price}\\ & K^{high} & = & \textrm{Higher-strike put strike price}\\ & P^{low}_0 & = & \textrm{Lower-strike put value at position opening (credit received)}\\ & P^{high}_0 & = & \textrm{Higher-strike put value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is unlimited, which occurs when the underlying price increases indefinitely.

The maximum loss is $K^{low} - K^{high} + P^{high}_0 - P^{low}_0 \times 2$, which occurs when the underlying price is exactly at the lower strike at expiry.

If the Option is American Option, there is a risk of early assignment on the contract you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	**Price ($)**	**Strike ($)**
Lower-Strike put	6.90	825.00
Higher-strike put	8.50	835.00
Underlying Equity at expiration	843.19	-

Therefore, the payoff is

$$\begin{array}{rcll} P^{low}_T & = & (K^{low} - S_T)^{+}\\ & = & (825.00-843.19)^{+}\\ & = & 0.00\\ P^{high}_T & = & (K^{high} - S_T)^{+}\\ & = & (835.00-843.19)^{+}\\ & = & 0.00\\ Payoff_T & = & (P^{high}_0 - P^{high}_T + P^{low}_T \times 2 - P^{low}_0 \times 2)\times m - fee\\ & = & (8.50 - 0.00 + 0.00\times2 - 6.90\times2)\times100-2.30\\ & = & -532.30\\ \end{array}$$

So, the strategy loses $532.30.

The following algorithm implements a long put backspread Option strategy:

```C#
public class BackspreadOptionStrategyAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);

        UniverseSettings.Asynchronous = true;
        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().PutSpread(20, 5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested ||
            !slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            return;
        }

        // Select the call Option contracts with the furthest expiry
        var expiry = chain.Max(x => x.Expiry);
        var puts = chain.Where(x => x.Expiry == expiry);
        if (puts.Count() == 0) return;

        // Select the strike prices from the remaining contracts
        var strikes = puts.Select(x => x.Strike).Distinct().OrderBy(x => x).ToList();
        if (strikes.Count < 2)
        {
            return;
        }

        var lowStrike = strikes[0];
        var highStrike = strikes[1];

        var optionStrategy = OptionStrategies.PutBackspread(_symbol, highStrike, lowStrike, expiry);
        Buy(optionStrategy, 1);
    }
}
```

Option Strategies

Short Put Backspread

Introduction

Short Put Backspread , consists of short 1 higher-strike put and short 2 lower-strike puts. It is a combination of a bear put spread and a short put with the same strike price as the lower-strike leg the put spread. All puts have the same underlying Equity and expiration date. This strategy profits from stable, consistent price of the underlying asset. For instance, the underlying price stays at its current price.

Implementation

Follow these steps to implement the short put backspread strategy:

1. In the `Initialize` method, set the start date, end date, cash, and Option universe . You can use the `PutSpread` helper method in option universe filtering, since a put backspread consists of the same contracts as a put spread.

```C#
private Symbol _symbol;

public override void Initialize()
{
    SetStartDate(2024, 9, 1);
    SetEndDate(2024, 12, 31);
    SetCash(1000000);

    UniverseSettings.Asynchronous = true;
    var option = AddOption("GOOG", Resolution.Minute);
    _symbol = option.Symbol;
    option.SetFilter(universe => universe.IncludeWeeklys().PutSpread(20, 5));
}
```

2. In the `OnData` method, select the expiration and strikes of the contracts in the strategy legs.

```C#
public override void OnData(Slice slice)
{
    if (Portfolio.Invested ||
        !slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    // Select the put Option contracts with the furthest expiry
    var expiry = chain.Max(x => x.Expiry);
    var puts = chain.Where(x => x.Expiry == expiry);
    if (puts.Count() == 0) return;

    // Select the strike prices from the remaining contracts
    var strikes = puts.Select(x => x.Strike).Distinct().OrderBy(x => x).ToList();
    if (strikes.Count < 2)
    {
        return;
    }

    var lowStrike = strikes[0];
    var highStrike = strikes[1];
```

3. In the `OnData` method, select the contracts and place the orders.

Approach A: Call the `OptionStrategies.ShortPutBackspread` method with the details of each leg and then pass the result to the `Buy` method.

```
C#
var optionStrategy = OptionStrategies.ShortPutBackspread(_symbol, highStrike, lowStrike, expiry);
Buy(optionStrategy, 1);
```

Approach B: Create a list of `Leg` objects and then call the Combo Market Order , Combo Limit Order , or Combo Leg Limit Order method.

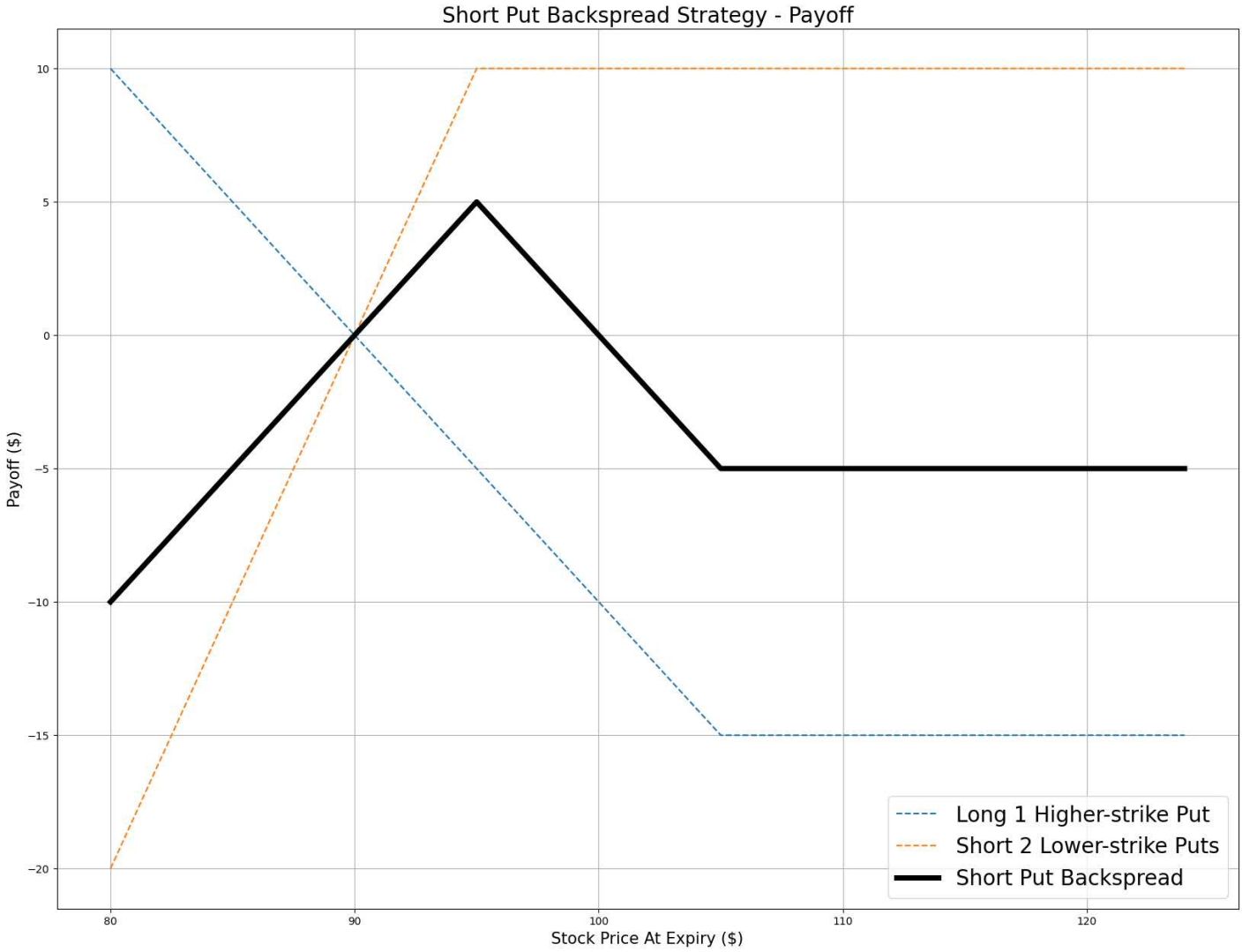
```
C#
var lowStrikePut = puts.Single(x => x.Strike == lowStrike);
var highStrikePut = puts.Single(x => x.Strike == highStrike);

var legs = new List<Leg>()
{
    Leg.Create(lowStrikePut.Symbol, -2),
    Leg.Create(highStrikePut.Symbol, 1)
};
ComboMarketOrder(legs, 1, true);
```

Strategy Payoff

The short put backspread is an unlimited-profit-limited-risk strategy. The payoff is

$$\begin{array}{rcll} P^{low}_T & = & (K^{low} - S_T)^{+}\\ P^{high}_T & = & (K^{high} - S_T)^{+}\\ Payoff_T & = & (P^{high}_T - P^{high}_0 - P^{low}_T \times 2 + P^{low}_0 \times 2)\times m - fee\\ \end{array} \begin{array}{rcll} \textrm{where} & P^{low}_T & = & \textrm{Lower-strike put value at time T}\\ & P^{high}_T & = & \textrm{Higher-strike put value at time T}\\ & S_T & = & \textrm{Underlying asset price at time T}\\ & K^{low} & = & \textrm{Lower-strike put strike price}\\ & K^{high} & = & \textrm{Higher-strike put strike price}\\ & P^{low}_0 & = & \textrm{Lower-strike put value at position opening (credit received)}\\ & P^{high}_0 & = & \textrm{Higher-strike put value at position opening (debit paid)}\\ & m & = & \textrm{Contract multiplier}\\ & T & = & \textrm{Time of expiration} \end{array}$$

The following chart shows the payoff at expiration:



The maximum profit is $K^{high} - K^{low} - P^{high}_0 + P^{low}_0 \times 2$, which occurs when the underlying price is exactly at the lower strike at expiry.

The maximum loss is $P^{low}_0 \times 2 - P^{high}_0 - S_T$, which occurs when the underlying price drops to zero.

If the Option is American Option, there is a risk of early assignment on the contract you sell.

Example

The following table shows the price details of the assets in the algorithm:

Asset	Price ($)	Strike ($)
Lower-Strike put	4.70	825.00
Higher-strike put	10.90	835.00
Underlying Equity at expiration	843.19	-

Therefore, the payoff is

$$\begin{array}{rcll} P^{low}_T & = & (K^{low} - S_T)^{+}\\ & = & (825.00-843.19)^{+}\\ & = & 0.00\\ P^{high}_T & = & (K^{high} - S_T)^{+}\\ & = & (835.00-843.19)^{+}\\ & = & 0.00\\ Payoff_T & = & (P^{high}_T - P^{high}_0 - P^{low}_T \times 2 + P^{low}_0 \times 2)\times m - fee\\ & = & (0 - 10.90 - 0.00\times2 + 4.70\times2)\times100-2.30\\ & = & -152.30\\ \end{array}$$

So, the strategy loses $152.30.

The following algorithm implements a short put backspread Option strategy:

```C#
public class BackspreadOptionStrategyAlgorithm : QCAlgorithm
{
    private Symbol _symbol;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);

        UniverseSettings.Asynchronous = true;
        var option = AddOption("GOOG", Resolution.Minute);
        _symbol = option.Symbol;
        option.SetFilter(universe => universe.IncludeWeeklys().PutSpread(20, 5));
    }

    public override void OnData(Slice slice)
    {
        if (Portfolio.Invested ||
            !slice.OptionChains.TryGetValue(_symbol, out var chain))
        {
            return;
        }

        // Select the call Option contracts with the furthest expiry
        var expiry = chain.Max(x => x.Expiry);
        var puts = chain.Where(x => x.Expiry == expiry);
        if (puts.Count() == 0) return;

        // Select the strike prices from the remaining contracts
        var strikes = puts.Select(x => x.Strike).Distinct().OrderBy(x => x).ToList();
        if (strikes.Count < 2)
        {
            return;
        }

        var lowStrike = strikes[0];
        var highStrike = strikes[1];

        var optionStrategy = OptionStrategies.ShortPutBackspread(_symbol, highStrike, lowStrike, expiry);
        Buy(optionStrategy, 1);
    }
}
```

Trading and Orders

Order Properties

Introduction

Order properties enable you to customize how the brokerage executes your orders. The `DefaultOrderProperties` of your algorithm sets the order properties for all of your orders. To adjust the order properties of an order, you can change the `DefaultOrderProperties` or pass an order properties object to the order method.

Time In Force

The `TimeInForce` property determines how long an order should remain open if it doesn't fill. This property doesn't apply to market orders since they usually fill immediately. Time in force is useful to automatically cancel old trades. The following table describes the available TimeInForce options:

Member	Example	Description
`GoodTilCanceled`	`TimeInForce.GoodTilCanceled`	Order is valid until filled (default)
`Day`	`TimeInForce.Day`	Order is valid until filled or the market closes
`GoodTilDate(DateTime expiry)`	`TimeInForce.GoodTilDate(new DateTime(2019, 6, 19, 12, 0, 0))`	Order is valid until filled or the specified expiration time

By default, orders remain open until they are canceled (`TimeInForce.GoodTilCanceled`). To update the value, set the `DefaultOrderProperties.TimeInForce` before you place an order or pass an `orderProperties` argument to the order method.

```C#
// Set a Limit Order to be good until market close
DefaultOrderProperties.TimeInForce = TimeInForce.Day;
LimitOrder("IBM", 100, 120);

// Set a Limit Order to be good until noon
LimitOrder("IBM", 100, 120, orderProperties: new OrderProperties
{
    TimeInForce = TimeInForce.GoodTilDate(new DateTime(2019, 6, 19, 12, 0, 0))
});
```

If you trade a market that's open 24 hours a day with daily data, `TimeInForce.Day` won't work because the order cancels at the market close time but your algorithm receives daily data at midnight.

Market on open (MOO) and market on close (MOC) orders don't support the `GoodTilDate` time in force. If you submit a MOO or MOC order with the `GoodTilDate` time in force, LEAN automatically adjusts the time in force to be `GoodTilCanceled` .

The brokerage you use may not support all of the `TimeInForce` options. To see the options that your brokerage supports, see the Orders section of the brokerage model documentation .

Brokerage-Specific Properties

Some brokerages support additional order properties so you can customize how your orders execute. Some examples include the following order properties:

- Financial Advisor group orders
- An `OutsideRegularTradingHours` property to let orders fill during pre-market and post-market trading hours
- A `PostOnly` property to force an order to only add liquidity to a market
- A `Hidden` property to make an order not show on the order book
- A `ReduceOnly` property to signal the order must only decrease your position size
- `FeeInBase` and `FeeInQuote` properties to set which currency you pay fees in for a Crypto trade

To view the order properties your brokerage supports, see the Orders section of the brokerage model documentation .

Tags

You can tag orders to aid your strategy development. Tags can be any string of up to 100 characters. Tags aren't part of the `OrderProperties` object, but they are a property of the `Order` class you can set. To set an order tag, pass it as an argument when you create the order or use the order update methods.

```
C#

// Tag an order on creation
var ticket = LimitOrder("SPY", 100, 221.05, tag: "Original tag");

// Update the tag with UpdateTag
ticket.UpdateTag("Updated tag");

// Update the tag with UpdateOrderFields
ticket.Update(new() { Tag = "Another tag" });
```

Examples

The following examples demonstrate some common practices for utilizing order properties.

Example 1: Extended Market Hour Orders

The following algorithm trades buys or sells SPY during pre-market to capture momentum in the overnight gap. It exits the positions at 15 minutes after the market opens. To trade in the extended market hours without stale fills, it set the respective order properties.

```C#
public class OrderPropertiesAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private decimal _lastPrice = 0m;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Add extended market hour SPY data for trading.
        _spy = AddEquity("SPY", extendedMarketHours: true).Symbol;

        // Add a Scheduled Event to trade 2 hours enter a position during pre-market.
        Schedule.On(DateRules.EveryDay(_spy), TimeRules.BeforeMarketOpen(_spy, 120), OpenPosition);
        // Add a Scheduled Event to exit the position 15 minutes after market open.
        Schedule.On(DateRules.EveryDay(_spy), TimeRules.AfterMarketOpen(_spy, 15), () => Liquidate());
        // Add a Scheduled Event to cache the daily close price so you can generate signals the next day.
        Schedule.On(DateRules.EveryDay(_spy), TimeRules.BeforeMarketClose(_spy), GetLastPrice);
        // Add a warm-up period so the algorithm trades on the first day in live mode.
        SetWarmUp(TimeSpan.FromDays(1));
    }

    private void OpenPosition()
    {
        if (_lastPrice > 0m)
        {
            var currentPrice = Securities[_spy].Price;

            // Define the order properties to trade in pre-market. The order will only valid for 3 minutes to avoid
            // stale fills.
            var orderProperties = new InteractiveBrokersOrderProperties
            {
                TimeInForce = TimeInForce.GoodTilDate(Time.AddMinutes(3)),
                OutsideRegularTradingHours = true
            };

            // Buy if the overnight direction is upward, indicating positive market sentiment.
            // Sell if the overnight direction is downward, indicating negative market sentiment.
            // Note that pre-market orders do not accept market order.
            LimitOrder(_spy, currentPrice > _lastPrice ? 10 : -10, currentPrice, orderProperties: orderProperties);
        }
    }

    private void GetLastPrice()
    {
        _lastPrice = Securities[_spy].Price;
    }
}
```

Example 2: Financial Advisor Managing Client Funds

The following algorithm forms an equal-weight portfolio at the start of each week with the 10 most liquid US Equities. It demonstrates how order properties can give financial advisors easy control over client funds.

C#

```
public class OrderPropertiesAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Define the order properties to trade with an FA group.
        DefaultOrderProperties = new InteractiveBrokersOrderProperties
        {
            FaGroup = "TestGroupEQ",        // FA group
            FaMethod = "NetLiq",            // Allocation by net liquidation
            Account = "FA123456"            // FA account
        };

        // Add a universe that selects the 10 most liquid US Equities at the start of each week.
        var spy = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
        UniverseSettings.Schedule.On(DateRules.WeekStart(spy));
        var universe = AddUniverse(Universe.Top(10));

        // Add a Scheduled Event to rebalance at the start of each week.
        Schedule.On(
            DateRules.WeekStart(spy),
            TimeRules.At(10, 0),
            // Form an equal-weighted portfolio of the universe constituents.
            () => SetHoldings(
                universe.Selected.Select(symbol => new PortfolioTarget(symbol, 0.1m)).ToList(),
                // Liquidate the assets that aren't in the universe anymore.
                liquidateExistingHoldings: true
            )
        );
    }
}
```

Trading and Orders

Order Events

Introduction

An `OrderEvent` object represents an update to the state of an order. As the state of your orders change, we notify your algorithm with `OrderEvent` objects through the `OnOrderEvent` and `OnAssignmentOrderEvent` event handlers.

Track Order Events

Each order generates events over its life as its status changes. Your algorithm receives these events through the `OnOrderEvent` and `OnAssignmentOrderEvent` methods. The `OnOrderEvent` event handler receives all order events. The `OnAssignmentOrderEvent` receives order events for Option assignments. The event handlers receive an `OrderEvent` object, which contains information about the order status.

```C#
public override void OnOrderEvent(OrderEvent orderEvent)
{
    var order = Transactions.GetOrderById(orderEvent.OrderId);
    if (orderEvent.Status == OrderStatus.Filled)
    {
        Debug($"{Time}: {order.Type}: {orderEvent}");
    }
}

public override void OnAssignmentOrderEvent(OrderEvent assignmentEvent)
{
    Log(assignmentEvent.ToString());
}
```

To get a list of all `OrderEvent` objects for an order, call the `OrderEvents` method of the order ticket .

```C#
var orderEvents = orderTicket.OrderEvents();
```

If you don't have the order ticket, get the order ticket from the TransactionManager .

Order States

Orders can have any of the following states:

Event Attributes

The `OnOrderEvent` and `OnAssignmentOrderEvent` event handlers in your algorithm receive `OrderEvent` objects, which have the following attributes:

Examples

The following examples demonstrate some common practices for using order events.

Example 1: Illiquid Stock Partial Fill

The following algorithm trades EMA cross on CARZ, an illiquid ETF. To realistically simulate the fill behavior, we set a fill model to partially fill the orders with at most 50% of the previous bar's volume per fill. We cancel the remaining open order after the

partial fill since we only trade on the updated information.

C#

```
public class OrderEventsAlgorithm : QCAlgorithm
{
    private Symbol _carz;
    private ExponentialMovingAverage _ema;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request CARZ data to feed indicator and trade.
        var equity = AddEquity("CARZ");
        _carz = equity.Symbol;
        // Set a custom partial fill model for the illiquid CARZ stock since it is more realistic.
        equity.SetFillModel(new CustomPartialFillModel(this));

        // Create EMA indicator to generate trade signals.
        _ema = EMA(_carz, 60, Resolution.Daily);
        // Warm up indicator for immediate readiness to use.
        WarmUpIndicator(_carz, _ema, Resolution.Daily);
    }

    public override void OnData(Slice slice)
    {
        if (slice.Bars.TryGetValue(_carz, out var bar))
        {
            // Trade EMA cross on CARZ for trend-following strategy.
            if (bar.Close > _ema && !Portfolio[_carz].IsLong)
            {
                SetHoldings(_carz, 0.5m);
            }
            else if (bar.Close < _ema && !Portfolio[_carz].IsShort)
            {
                SetHoldings(_carz, -0.5m);
            }
        }
    }

    public override void OnOrderEvent(OrderEvent orderEvent)
    {
        // If an order is only partially filled, we cancel the rest to avoid trade on non-updated information.
        if (orderEvent.Status == OrderStatus.PartiallyFilled)
        {
            Transactions.CancelOpenOrders();
        }
    }

    /// Implements a custom fill model that partially fills each order with a ratio of the previous trade bar.
    private class CustomPartialFillModel : FillModel
    {
        private readonly QCAlgorithm _algorithm;
        private readonly Dictionary<int, decimal> _absoluteRemainingByOrderId;
        // Save the ratio of the volume of the previous bar to fill the order.
        private decimal _ratio;

        public CustomPartialFillModel(QCAlgorithm algorithm, decimal ratio = 0.5m)
            : base()
        {
            _algorithm = algorithm;
            _absoluteRemainingByOrderId = new Dictionary<int, decimal>();
            _ratio = ratio;
        }

        public override OrderEvent MarketFill(Security asset, MarketOrder order)
        {
            decimal absoluteRemaining;
            if (!_absoluteRemainingByOrderId.TryGetValue(order.Id, out absoluteRemaining))
            {
                absoluteRemaining = order.AbsoluteQuantity;
            }

            var fill = base.MarketFill(asset, order);

            // Partially filled each order with at most 50% of the previous bar.
            fill.FillQuantity = Math.Sign(order.Quantity) * asset.Volume * _ratio;
            if (Math.Min(Math.Abs(fill.FillQuantity), absoluteRemaining) == absoluteRemaining)
            {
```

```
                fill.FillQuantity = Math.Sign(order.Quantity) * absoluteRemaining;
                fill.Status = OrderStatus.Filled;
                _absoluteRemainingByOrderId.Remove(order.Id);
            }
            else
            {
                fill.Status = OrderStatus.PartiallyFilled;
                // Save the remaining quantity after it is partially filled.
                _absoluteRemainingByOrderId[order.Id] = absoluteRemaining - Math.Abs(fill.FillQuantity);
                var price = fill.FillPrice;
            }
            return fill;
        }
    }
}
```

Example 2: Price Actions

The following algorithm saves the trailing 3 `TradeBar` objects into a `RollingWindow` . When it identifies a volume contraction breakout price action pattern on the SPY, it buys to ride on the capital inflow. To exit positions, it places a 2% take profit and 1% stop loss order in the `OnOrderEvent` method.

C#

```
public class RollingWindowAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    // Set up a rolling window to hold the last 3 trade bars for price action detection as the trade signal.
    private RollingWindow<TradeBar> _windows = new(3);

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Add SPY data for signal generation and trading.
        _spy = AddEquity("SPY", Resolution.Minute).Symbol;

        // Warm up the rolling window.
        var history = History<TradeBar>(_spy, 3, Resolution.Minute);
        foreach (var bar in history)
        {
            _windows.Add(bar);
        }
    }

    public override void OnData(Slice slice)
    {
        if (slice.Bars.TryGetValue(_spy, out var bar))
        {
            // Trade the price action if the previous bars fulfill a contraction breakout.
            if (ContractionAction() && BreakoutAction(bar.Close))
            {
                SetHoldings(_spy, 0.5m);
            }

            // Add the current bar to the window.
            _windows.Add(bar);
        }
    }

    private bool ContractionAction()
    {
        // We trade contraction type price action, where the buying preesure is increasing.
        // 1. The last 3 bars are green.
        // 2. The price is increasing in trend.
        // 3. The trading Volume is increasing as well.
        // 4. The range of the bars are decreasing.
        return _windows[2].Close > _windows[2].Open &&
            _windows[1].Close > _windows[1].Open &&
            _windows[0].Close > _windows[0].Open &&
            _windows[0].Close > _windows[1].Close && _windows[1].Close > _windows[2].Close &&
            _windows[0].Volume > _windows[1].Volume && _windows[1].Volume > _windows[2].Volume &&
            _windows[2].Close - _windows[2].Open > _windows[1].Close - _windows[1].Open &&
            _windows[1].Close - _windows[1].Open > _windows[0].Close - _windows[0].Open;
    }
```

```
    private bool BreakoutAction(decimal currentPrice)
    {
        // Trade breakout from contraction: the breakout should be much greater than the contracted range of the last
bar.
        return currentPrice - _windows[0].Close > (_windows[0].Close - _windows[0].Open) * 2m;
    }

    public override void OnOrderEvent(OrderEvent orderEvent)
    {
        if (orderEvent.Status == OrderStatus.Filled)
        {
            if (orderEvent.Ticket.OrderType == OrderType.Market)
            {
                // Stop loss order at 1%.
                var stopPrice = orderEvent.FillQuantity > 0m ? orderEvent.FillPrice * 0.99m : orderEvent.FillPrice *
1.01m;
                StopMarketOrder(_spy, -Portfolio[_spy].Quantity, stopPrice);
                // Take profit order at 2%.
                var takeProfitPrice = orderEvent.FillQuantity > 0m ? orderEvent.FillPrice * 1.02m :
orderEvent.FillPrice * 0.98m;
                LimitOrder(_spy, -Portfolio[_spy].Quantity, takeProfitPrice);
            }
            else if (orderEvent.Ticket.OrderType == OrderType.StopMarket || orderEvent.Ticket.OrderType ==
OrderType.Limit)
            {
                // Cancel open orders if the stop loss or take profit order fills.
                Transactions.CancelOpenOrders();
            }
        }
    }
}
```

Trading and Orders

Order Errors

Introduction

If an error occurs with one of your orders, LEAN logs the error message and stops executing.

Types of Order Errors

Errors can occur when you place orders or after you place them.

Errors at Order Submission

The following errors can occur when you submit an order:

- `AlgorithmWarmingUp`
- `BrokerageFailedToSubmitOrder`
- `BrokerageModelRefusedToSubmitOrder`
- `ConversionRateZero`
- `EuropeanOptionNotExpiredOnExercise`
- `ExceededMaximumOrders`
- `ExceedsShortableQuantity`
- `ExchangeNotOpen`
- `ForexBaseAndQuoteCurrenciesRequired`
- `ForexConversionRateZero`
- `InsufficientBuyingPower`
- `MarketOnCloseOrderTooLate`
- `MissingSecurity`
- `NonExercisableSecurity`
- `NonTradableSecurity`
- `OptionOrderOnStockSplit`
- `OrderAlreadyExisits`
- `OrderQuantityLessThanLotSize`
- `OrderQuantityZero`
- `PreOrderChecksError`
- `QuoteCurrencyRequired`
- `SecurityHasNoData`
- `UnsupportedRequestType`

Errors After Order Submission

The following errors can occur for active orders:

- `BrokerageFailedToUpdateOrder`
- `BrokerageModelRefusedToUpdateOrder`
- `InvalidNewOrderStatus`
- `InvalidOrderStatus`

- `InvalidRequest`
- `RequestCanceled`
- `UnableToFindOrder`

Common Errors

There are a few common order errors you may experience.

Why is my order converted to a market on open order?

If you place a market order when the market is closed, LEAN automatically converts the order into market on open order. This most commonly happens when you use daily or hourly data, which your algorithm can receive when the market is closed. With the `DailyPreciseEndTime` setting enabled, your algorithm receives daily bars and the last hourly bar of each day at 4 PM Eastern Time (ET). To avoid LEAN from converting your market order to market on open orders, submit your market orders when the market is open. To check if the market is open, call the `IsMarketOpen` method.

```C#
// Check the market status before placing an order to avoid unintended market-on-open trades
if (IsMarketOpen(_symbol))
{
    MarketOrder(symbol, quantity);
}
```

Why am I seeing the "stale price" warning?

Stale fills occur when you fill an order with price data that is timestamped an hour or more into the past. Stale fills usually only occur if you trade illiquid assets or if your algorithm uses daily data but you trade intraday with Scheduled Events. If your order is filled with stale data, the fill price may not be realistic. The pre-built fill models can only fill market orders with stale data. To adjust the length of time that needs to pass before an order is considered stale, set the `StalePriceTimeSpan` setting.

```C#
public override void Initialize()
{
    // Adjust the stale price time span to be 10 minutes.
    Settings.StalePriceTimeSpan = TimeSpan.FromMinutes(10);
}
```

Why do I get "Backtest Handled Error: The security with symbol '/ES' is marked as non-tradable"?

This error occurs when you place an order for a continuous Futures contract, which isn't a tradable security. To fix the issue, place the order for a specific Futures contract. To access the currently selected contract in the continuous contract series, use the `Mapped` property of the `Future` object.

Why do I get "Backtest Handled Error: Order Error: ids: [1], Insufficient buying power to complete orders" from a Crypto order?

When you place Crypto trades, ensure you have a sufficient balance of the base or quote currency before each trade. If you hold multiple assets and you want to put all of your capital into BTCUSDT, you need to first convert all your non-BTC assets into USDT and then purchase BTCUSDT.

By default, the account currency is USD and your starting cash is 100,000. Some Crypto exchanges don't support fiat currencies, which means you can't convert the 100,000 USD to the pair's quote currency . In this case, set your account currency to the quote currency with a positive quantity.

```C#
SetAccountCurrency("USDT", 1000);   // Set the account currency to Tether and its quantity to 1,000 USDT
```

Error Code Reference

The `OrderError` enumeration has the following members:

Order Response Error Reference

The following sections explain why each `OrderResponseErrorCode` occurs and how to avoid it.

None

The `OrderResponseErrorCode.None` (0) error means there is no order response error.

Processing Error

The `OrderResponseErrorCode.ProcessingError` (-1) error occurs in the following situations:

- When you submit a new order, but LEAN throws an exception while checking if you have sufficient buying power for the order.
- When you try to update or cancel an order, but LEAN throws an exception.

To investigate this order response error further, see the `HandleSubmitOrderRequest` , `UpdateOrder` , and `CancelOrder` methods of the BrokerageTransactionHandler in the LEAN GitHub repository.

Order Already Exists

The `OrderResponseErrorCode.OrderAlreadyExists` (-2) error occurs when you submit a new order but you already have an open order or a completed order with the same order ID. This order response error usually comes from a concurrency issue.

To avoid this order response, don't place two asynchronous orders at the same time.

Insufficient Buying Power

The `OrderResponseErrorCode.InsufficientBuyingPower` (-3) error occurs when you place an order but the buying power model determines you can't afford it.

To avoid this order response error, ensure you have enough margin remaining to cover the initial margin requirements of the order before placing it.

This error also commonly occurs when you place a market on open order with daily data. If you place the order with `SetHoldings` or use `CalculateOrderQuantity` to determine the order quantity, LEAN calculates the order quantity based on the market close price. If the open price on the following day makes your order more expensive, then you may have insufficient buying power. To avoid the order response error in this case, either use intraday data and place trades when the market is open or adjust your buying power buffer .

```C#
Settings.FreePortfolioValuePercentage = 0.05m;
```

Brokerage Model Refused to Submit Order

The `OrderResponseErrorCode.BrokerageModelRefusedToSubmitOrder` (-4) error occurs when you place an order but the brokerage model determines it's invalid. The brokerage model usually checks your order meets the following requirements before sending it to the brokerage:

- Supported security types
- Supported order types and their respective requirements
- Supported time in force options
- The order size is larger than the minimum order size

Each brokerage model can have additional order requirements that the brokerage declares. To avoid this order response error, see the Orders section of the brokerage model documentation .

To investigate this order response error further, see the `CanSubmitOrder` method definition of your brokerage model . This order response error occurs when the `CanSubmitOrder` method returns `false` .

Brokerage Failed to Submit Order

The `OrderResponseErrorCode.BrokerageFailedToSubmitOrder` (-5) error occurs when you place an order but the brokerage implementation fails to submit the order to your brokerage.

To investigate this order response error further, see the `PlaceOrder` method definition of your brokerage or the BacktestingBrokerage in the LEAN GitHub repository. This order response error occurs when the `PlaceOrder` method throws an error or returns `false` .

Brokerage Failed to Update Order

The `OrderResponseErrorCode.BrokerageFailedToUpdateOrder` (-6) error occurs when you try to update an order but the brokerage implementation fails to submit the order update request to your brokerage.

To avoid this order response error, see the Orders section of the brokerage model documentation .

To investigate this order response error further, see the `UpdateOrder` method definition of your brokerage or the BacktestingBrokerage in the LEAN GitHub repository. This order response error occurs when the `UpdateOrder` method throws an error or returns `false` .

Brokerage Failed to Cancel Order

The `OrderResponseErrorCode.BrokerageFailedToCancelOrder` (-8) error occurs when you try to cancel an order but the brokerage implementation fails to submit the cancel request to your brokerage.

To investigate this order response error further, see the `CancelOrder` method definition of your brokerage or the BacktestingBrokerage in the LEAN GitHub repository. This order response error occurs when `CancelOrder` method throws an error or returns `false` .

Invalid Order Status

The `OrderResponseErrorCode.InvalidOrderStatus` (-9) error occurs when you try to update or cancel an order but the order is already complete. An order is complete if it has `OrderStatus.Filled` , `OrderStatus.Canceled` , or `OrderStatus.Invalid` .

To avoid this order response error, check the `Status` of an order ticket or order event before you update or cancel the order.

```C#
if (!_orderTicket.Status.IsClosed())
{
    _orderTicket.Cancel();
}
```

Unable to Find Order

The `OrderResponseErrorCode.UnableToFindOrder` (-10) error occurs when you try to place, update, or cancel an order, but the `BrokerageTransactionHandler` doesn't have a record of the order ID.

To investigate this order response error further, see BrokerageTransactionHandler.cs in the LEAN GitHub repository. This order response error occurs when the `BrokerageTransactionHandler` can't find the order ID in it's `_completeOrders` or `_completeOrderTickets` dictionaries.

Order Quantity Zero

The `OrderResponseErrorCode.OrderQuantityZero` (-11) error occurs when you place an order that has zero quantity or when you update an order to have a zero quantity. This error commonly occurs if you use the SetHoldings method but the portfolio weight you provide to the method is too small to translate into a non-zero order quantity.

To avoid this order response error, check if the quantity of the order is non-zero before you place the order. If you use the `SetHoldings` method, replace it with a combination of the CalculateOrderQuantity and MarketOrder methods.

```
C#

var quantity = CalculateOrderQuantity(_symbol, 0.05);
if (quantity != 0)
{
    MarketOrder(_symbol, quantity);
}
```

Unsupported Request Type

The `OrderResponseErrorCode.UnsupportedRequestType` (-12) error occurs in the following situations:

- When you try to exercise an Option contract for which you hold a short position
- When you try to exercise more Option contracts than you hold

To avoid this order response error, check the quantity of your holdings before you try to exercise an Option contract.

```
C#

var holdingQuantity = Portfolio[_contractSymbol].Quantity;
if (holdingQuantity > 0)
{
    ExerciseOption(_contractSymbol, Math.Max(holdingQuantity, exerciseQuantity));
}
```

Missing Security

The `OrderResponseErrorCode.MissingSecurity` (-14) error occurs when you place an order for a security but you don't have a subscription for the security in your algorithm.

To avoid this order response error, create a subscription for each security you want to trade. To create subscriptions, see the Requesting Data page of the documentation for each asset class .

Exchange Not Open

The `OrderResponseErrorCode.ExchangeNotOpen` (-15) error occurs in the following situations:

- When you try to exercise an Option while the exchange is not open

  To avoid the order response error in this case, check if the exchange is open before you exercise an Option contract.

```
if (IsMarketOpen(_contractSymbol))
{
    ExerciseOption(_contractSymbol, quantity);
}
```

- When you try to place a market on open order for a Futures contract or a Future Option contract

Security Price Zero

The `OrderResponseErrorCode.SecurityPriceZero` (-16) error occurs when you place an order or exercise an Option contract while the security price is 0. The security price can be 0 for the following reasons:

- The data is missing

  Investigate if it's a data issue. If it is a data issue, report it .

- The algorithm hasn't received data for the security yet

  If you subscribe to a security and place an order for the security in the same time step , you'll get this error. To avoid this order response error, enable the `SeedInitialPrices` setting to seed new assets with their last known price.

  ```
  Settings.SeedInitialPrices = true;
  ```

Forex Base and Quote Currencies Required

The `OrderResponseErrorCode.ForexBaseAndQuoteCurrenciesRequired` (-17) error occurs when you place a trade for a Forex or Crypto pair but you don't have the base currency and quote currency in your cash book . This error should never occur. If it does, create a bug report.

Forex Conversion Rate Zero

The `OrderResponseErrorCode.ForexConversionRateZero` (-18) error occurs when you place a trade for a Forex or Crypto pair and LEAN can't convert the value of the base currency to your account currency. This error usually indicates a lack of data. Investigate the data and if there is some missing, report it .

Security Has No Data

The `OrderResponseErrorCode.SecurityHasNoData` (-19) error occurs when you place an order for a security before your algorithm receives any data for it. If you subscribe to a security and place an order for the security in the same time step , you'll get this error. To avoid this order response error, enable the `SeedInitialPrices` setting to seed new assets with their last known price.

```
Settings.SeedInitialPrices = true;
```

Exceeded Maximum Orders

The `OrderResponseErrorCode.ExceededMaximumOrders` (-20) error occurs when exceed your order quota in a backtest. The number of orders you can place in a single backtest depends on the tier of your organization . The following table shows the number of orders you can place on each tier:

Tier	Orders Quota
Free	10K
Quant Researcher	10M
Team	Unlimited
Trading Firm	Unlimited
Institution	Unlimited

To avoid this order response error, reduce the number of orders in your backtest or upgrade your organization .

Market on Close Order Too Late

The `OrderResponseErrorCode.MarketOnCloseOrderTooLate` (-21) error occurs when you try to place a market on close (MOC) order too early in the trading day.

To avoid this order response error, place the MOC order closer to the market close or adjust the submission time buffer. By default, you must place MOC orders at least 15.5 minutes before the close, but some exchanges let you submit them closer to the market closing time. To adjust the buffer period that's required, set the `MarketOnCloseOrder.SubmissionTimeBuffer` property.

```
C#
Orders.MarketOnCloseOrder.SubmissionTimeBuffer = TimeSpan.FromMinutes(10);
```

Invalid Request

The `OrderResponseErrorCode.InvalidRequest` (-22) error occurs when you try to cancel an order multiple times.

To avoid this order response error, only try to cancel an order one time.

Request Canceled

The `OrderResponseErrorCode.RequestCanceled` (-23) error occurs when you try to cancel an order multiple times.

To avoid this order response error, only try to cancel an order one time.

Algorithm Warming Up

The `OrderResponseErrorCode.AlgorithmWarmingUp` (-24) error occurs in the following situations:

- When you try to place, update, or cancel an order during the warm-up period
- When the Option assignment simulator assigns you to an Option during the warm-up period

To avoid this order response error, only manage orders after the warm-up period ends. To avoid trading during the warm-up period, add an `IsWarmingUp` guard to the top of the `OnData` method.

```
C#
if (IsWarmingUp) return;
```

Brokerage Model Refused to Update Order

The `OrderResponseErrorCode.BrokerageModelRefusedToUpdateOrder` (-25) error occurs in backtests when you try to update

an order in a way that the brokerage model doesn't support.

To avoid this issue, see the Orders section of the brokerage model documentation to check its order requirements.

To investigate this order response error further, see the `CanUpdateOrder` method definition of your brokerage model .

Quote Currency Required

The `OrderResponseErrorCode.QuoteCurrencyRequired` (-26) error occurs when you place an order for a Forex or Crypto pair and don't have the quote currency of the pair in your cash book . This error should never occur. If it does, create a bug report.

Conversion Rate Zero

The `OrderResponseErrorCode.ConversionRateZero` (-27) error occurs when you place an order for a Forex or Crypto pair and LEAN can't convert the value of the quote currency in the pair to your account currency. This order response error usually indicates a lack of data. Investigate the data and if there is data missing, report it .

Non-Tradable Security

The `OrderResponseErrorCode.NonTradableSecurity` (-28) error occurs when you place an order for a security that's not tradable . To avoid this order response error, check if a security is tradable before you trade it.

```
C#
if (Securities[_symbol].IsTradable)
{
    MarketOrder(_symbol, quantity);
}
```

Non-Exercisable Security

The `OrderResponseErrorCode.NonExercisableSecurity` (-29) error occurs when you call the ExerciseOption method with a `Symbol` that doesn't reference an Option contract.

Order Quantity Less Than Lot Size

The `OrderResponseErrorCode.OrderQuantityLessThanLotSize` (-30) error occurs when you place an order with a quantity that's less than the lot size of the security.

To avoid this order response error, check if the order quantity is greater than or equal to the security lot size before you place an order.

```
C#
var lotSize = Securities[_symbol].SymbolProperties.LotSize;
if (quantity >= lotSize)
{
    MarketOrder(_symbol, quantity);
}
```

Exceeds Shortable Quantity

The `OrderResponseErrorCode.ExceedsShortableQuantity` (-31) error occurs when you place an order to short a security but the shortable provider of the brokerage model states there isn't enough shares to borrow. For a full example of this error, clone and run this backtest .

To avoid this order response error, check if there are enough shares available before you place an order to short a security.

```
if (quantityToBorrow <= ShortableQuantity(_symbol))
{
    MarketOrder(_symbol, -quantityToBorrow);
}
```

Invalid New Order Status

The `OrderResponseErrorCode.InvalidNewOrderStatus` (-32) error occurs in live trading when you try to update or cancel an order while it still has `OrderStatus.New` status.

To avoid this order response error, check the `Status` property of the order ticket or order event before you update or cancel an order.

```
if (_orderTicket.Status != OrderStatus.New)
{
    _orderTicket.Cancel();
}
```

European Option Not Expired on Exercise

The `OrderResponseErrorCode.EuropeanOptionNotExpiredOnExercise` (-33) error occurs when you try to exercise a European Option contract before its expiry date.

To avoid this order response error, check the type and expiry date of the contract before you exercise it.

```
if (_contractSymbol.ID.OptionStyle == OptionStyle.European && _contractSymbol.ID.Date == Time.Date)
{
    ExerciseOption(_contractSymbol, quantity);
}
```

Option Order on Stock Split

The `OrderResponseErrorCode.OptionOrderOnStockSplit` (-34) error occurs when you try to submit an order for an Equity Option contract when the current time slice contains a split for the underlying Equity.

To avoid this order response error, check if the time slice has a split event for the underlying Equity of the contract before you place an order for the contract.

```
if (!slice.Splits.ContainsKey(_contractSymbol.Underlying))
{
    MarketOrder(_contractSymbol, quantity);
}
```

Market on Open Not Allowed During Regular Hours

The `OrderResponseErrorCode.MarketOnOpenNotAllowedDuringRegularHours` (-35) error occurs when you try to place a market on open order for an asset when it's during regular trading hours.

To avoid this order response error, place the order when the market is closed.

```C#
if (!IsMarketOpen(_symbol))
{
    MarketOnOpenOrder(_symbol, quantity);
}
```

Warnings

The following sections explain why each some warnings occur and how to avoid them.

Limit Price Exceeds Precision Requirements

Limit prices must be a multiple of the security's `MinimumPriceVariation` property . Otherwise, LEAN rounds the limit price for you and throws the following warning message:

Backtest Handled Error: Warning: To meet brokerage precision requirements, order LimitPrice was rounded to 6136.00 from 6136.125

To avoid the warning, round the limit price to the nearest value that respects the `MinimumPriceVariation` before you place the order.

```C#
private decimal AdjustLimitPrice(Security security, decimal price, bool roundUp = true)
{
    decimal pip = security.PriceVariationModel.GetMinimumPriceVariation(
        new GetMinimumPriceVariationParameters(security, price)
    );
    return ((int)(price / pip) + (roundUp ? 1 : 0)) * pip;
}
```

Examples

The following examples demonstrate common practices for handling order errors.

Example 1: Pre-Open Order Error

The following algorithm simulate extended market hour trading on Tradier Brokerage . We place a market-on-open order at 7:30am and expect the order cannot submit to the broker since it is not supported, resulting into an error.

C#

```
public class OrderErrorsAlgorithm : QCAlgorithm
{
    private Symbol _spy;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 5);
        // Simulate Tradier brokerage, which does not support the market-on-open orders.
        SetBrokerageModel(BrokerageName.TradierBrokerage, AccountType.Cash);

        // Request extended market hour SPY data for trading.
        _spy = AddEquity("SPY", extendedMarketHours: true).Symbol;

        // Set a scheduled event to trade 2 hours pre-open to place the market-on-open-order.
        Schedule.On(
            DateRules.EveryDay(_spy),
            TimeRules.BeforeMarketOpen(_spy, 120),
            OpenPosition
        );
    }

    private void OpenPosition()
    {
        // Buy on market open will result in an error since Tradier does not support it.
        MarketOnOpenOrder(_spy, 10);
    }

    // It will be triggered on live trading.
    public override void OnBrokerageMessage(BrokerageMessageEvent message)
    {
        if (message.Type == BrokerageMessageType.Error)
        {
            Log($"{Time}: {message.Type}: Message: {message.Message}");
        }
    }
}
```

Trading and Orders

Trade Statistics

Introduction

The `TradeBuilder` tracks the trades of your algorithm and calculates some statistics. Some of these statistics are a function of the risk free interest rate and the number of trading days per year . You can adjust how the `TradeBuilder` defines a trade and how it matches offsetting order fills.

Set Trade Builder

To set the `TradeBuilder` , in the `Initialize` method, call the `SetTradeBuilder` method.

```
C#
SetTradeBuilder(new TradeBuilder(groupingMethod, matchingMethod));
```

The following table describes the arguments the `TradeBuilder` constructor accepts:

Argument	Data Type	Description	Default Value
`groupingMethod`	`FillGroupingMethod`	The method used to group order fills into trades	
`matchingMethod`	`FillMatchingMethod`	The method used to match offsetting order fills	

The `FillGroupingMethod` enumeration has the following members:

The `FillMatchingMethod` enumeration has the following members:

Default Behavior

The default `TradeBuilder` uses `FillGroupingMethod.FillToFill` and `FillMatchingMethod.FIFO` .

Check Open Positions

To check if you have a position open for a security as defined by the `FillGroupingMethod` , call the `HasOpenPosition` method.

```
C#
// Check if the trade builder has recorded any active trades (unclosed positions)
// for a security.
var hasOpenPosition = TradeBuilder.HasOpenPosition(_symbol);
```

Get Closed Trades

To get your closed trades, use the `ClosedTrades` property.

```
C#
var trades = TradeBuilder.ClosedTrades;
```

This property returns a list of `Trade` objects, which have the following attributes:

Examples

The following examples demonstrate some common practices for making use of trade statistics.

Example 1: Kelly Criterion Sizing Using Previous Trades

The following algorithm uses the previous trades to update the win probability and expected return of each trade of the EMA cross strategy of SPY, by the concept of Bayes Theorem. The updated figures are used for position sizing through Kelly Criterion.

```C#
public class TradeStatisticsAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private ExponentialMovingAverage _ema;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 12);
        SetEndDate(2024, 10, 1);
        SetCash(1000000);

        // Request SPY data to trade.
        _spy = AddEquity("SPY").Symbol;
        // Create an EMA indicator to generate trade signals.
        _ema = EMA(_spy, 20, Resolution.Daily);
        // Warm-up indicator for immediate readiness.
        WarmUpIndicator(_spy, _ema, Resolution.Daily);

        // Set up a trade builder to track the trade statistics; we are interested in open-to-close round trade.
        SetTradeBuilder(new TradeBuilder(FillGroupingMethod.FlatToFlat, FillMatchingMethod.FIFO));
    }

    public override void OnData(Slice slice)
    {
        if (slice.Bars.TryGetValue(_spy, out var bar))
        {
            // Trend-following strategy using price and EMA.
            // If the price is above EMA, SPY is in an uptrend, and we buy it.
            var sign = 0m;
            if (bar.Close > _ema && !Portfolio[_spy].IsLong)
            {
                sign = 1m;
            }
            else if (bar.Close < _ema && !Portfolio[_spy].IsShort)
            {
                sign = -1m;
            }
            else
            {
                return;
            }

            var size = 1m;
            var trades = TradeBuilder.ClosedTrades;

            if (trades.Count > 4)
            {
                // Use the trade builder to obtain the win rate and % return of the last five trades to calculate
position size.
                var lastFiveTrades = trades.OrderBy(x => x.ExitTime).TakeLast(5);
                var probWin = lastFiveTrades.Count(x => x.IsWin) / 5m;
                var winSize = lastFiveTrades.Average(x => x.ProfitLoss / x.EntryPrice);
                // Use the Kelly Criterion to calculate the order size.
                size = probWin - (1 - probWin) / winSize;
            }

            SetHoldings(_spy, sign * size);
        }
    }
}
```

Trading and Orders

Trading Calendar

Introduction

The `TradingCalendar` contains a variety of events relevant to the assets in your algorithm.

Trading Day Properties

`TradingDay` objects represent the trading events of a single day. They have the following properties:

Get a Single Trading Day

To get the trading events of the current day, call the `GetTradingDay` method with no arguments.

```C#
var tradingDay = TradingCalendar.GetTradingDay();
```

To get the trading events of a specific day, pass a `DateTime` object to the `GetTradingDay` method.

```C#
var tradingDay = TradingCalendar.GetTradingDay(new DateTime(2022, 6, 1));
```

Get Trading Days

To get all the trading events across a range of dates, pass start and end `DateTime` objects to the `GetTradingDays` method.

```C#
var startDate = new DateTime(2022, 6, 1);
var endDate = new DateTime(2022, 7, 1);
var tradingDays = TradingCalendar.GetTradingDays(startDate, endDate);
```

To get specific trading events across a range of dates, pass a `TradingDayType` enumeration member and the dates to the `GetDaysByType` method.

```C#
var tradingDays = TradingCalendar.GetDaysByType(TradingDayType.OptionExpiration, startDate, endDate);
```

The `TradingDayType` enumeration has the following members:

Trading and Orders

Financial Advisors

Introduction

Financial Advisor accounts enable certified professionals to use a single trading algorithm to manage several client accounts. Our Interactive Brokers integration enables you to place FA group orders if your IB account code starts with F, FA, or I.

Group Routing

To place trades using a subset of client accounts, create Account Groups in Trader Workstation and then define the `InteractiveBrokersOrderProperties` when you create orders.

```C#
DefaultOrderProperties = new InteractiveBrokersOrderProperties
{
    FaGroup = "TestGroupEQ",
    FaMethod = "Equal",
    Account = "DU123456"
};
```

SecurityHolding objects aggregate your positions across all the account groups. If you have two groups where group A has 10 shares of SPY and group B has -10 shares of SPY, then `Portfolio["SPY"].Quantity` is zero.

Allocation Methods

LEAN supports several allocation methods for FA group orders. If you intend to use the same group allocation method for every order, set the `DefaultOrderProperties` of your algorithm, which sets the order properties for all of your orders.

```C#
public override void Initialize()
{
    // Set the default order properties
    DefaultOrderProperties = new InteractiveBrokersOrderProperties()
    {
        FaGroup = "TestGroupEQ",
        FaMethod = "Equal",
        Account = "DU123456"
    };
}

public override void OnData(Slice slice)
{
    // Use default order order properties
    LimitOrder(_symbol, quantity, limitPrice);
}
```

To adjust the order properties of an order, change the `DefaultOrderProperties` or pass an order properties object to the order method . The following sections explain the FA group allocation methods.

Equal

This group allocation method distributes shares equally between all accounts in the group. When you use this method, you need to specify an order quantity.

For example, say your Account Group includes four accounts and you place an order to buy 400 shares of a stock. In this case,

each account receives 100 shares. If your Account Group includes six accounts, each account receives 66 shares, and then 1 share is allocated to each account until all are distributed. After you submit the order, your algorithm receives order events to track the order progress. When all the shares are bought, the order status is `OrderStatus.Filled`. If one of the accounts in the group can't afford 10 of the shares it needs to buy, 10 shares are cancelled and you'll only end up buying 390 shares in total.

```
C#
LimitOrder(
    _symbol, quantity, limitPrice,
    orderProperties: new InteractiveBrokersOrderProperties
    {
        FaMethod = "Equal"
    }
);
```

Net Liquidation Value

This group allocation method distributes shares based on the net liquidation value of each account. The system calculates ratios based on the net liquidation value in each account and allocates shares based on these ratios. When you use this method, you need to specify an order quantity.

For example, say your account group includes three accounts, A, B and C with Net Liquidation values of 25,000, 50,000 and $100,000, respectively. In this case, the system calculates a ratio of 1:2:4. If you place an order for 700 shares of a stock, it allocates 100 shares to Client A, 200 shares to Client B, and 400 shares to Client C.

```
C#
LimitOrder(_symbol, quantity, limitPrice,
    orderProperties: new InteractiveBrokersOrderProperties
    {
        FaMethod = "NetLiq"
    }
);
```

Available Equity

This group allocation method distributes shares based on the amount of available equity in each account. The system calculates ratios based on the available equity in each account and allocates shares based on these ratios. When you use this method, you need to specify an order quantity.

For example, say your account group includes three accounts, A, B and C with available equity of 25,000, 50,000 and $100,000, respectively. In this case, the system calculates a ratio of 1:2:4. If you place an order for 700 shares of a stock, it allocates 100 shares to Client A, 200 shares to Client B, and 400 shares to Client C.

```
C#
LimitOrder(_symbol, quantity, limitPrice,
    orderProperties: new InteractiveBrokersOrderProperties
    {
        FaMethod = "AvailableEquity"
    }
);
```

Subscription Requirements

To use FA group orders through our Interactive Brokers integration, you need to connect as a member of a Trading Firm and Institution organization. If you aren't currently on either of these tiers, upgrade your organization .

Reality Modeling

Reality Modeling

Key Concepts

Introduction

Reality models make backtests as realistic as possible to how the strategy would perform in live trading. These reality models model the behavior of things like the portfolio, brokerage, fills, slippage, options, and strategy capacity . Some reality models are set on a security basis and some are set at the portfolio level. The default models assume you trade highly liquid assets. If you trade high volumes or on illiquid assets, you should create custom reality models to be more realistic.

Security Level Models

LEAN's philosophy is to make the models customizable per security as much as possible. The following models are security-level models:

- Fill
- Slippage
- Fee
- Brokerages
- Buying Power
- Settlement
- Short availability
- Option models
  - Pricing
  - Volatility
  - Exercise
  - Assignment
- Margin Interest Rate
- Dividend Yield

To set a security-level reality model, call the set reality model method on the `Security` object. To get the correct method, see the preceding documentation page for each type of model.

```C#
// Set IBM to have a constant $1 transaction fee
Securities["IBM"].SetFeeModel(new ConstantFeeModel(1));
```

You can also set the security-specific models inside a security initializer .

```C#
public class AddSecurityInitializerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the security initializer to set models of assets.
        AddSecurityInitializer(CustomSecurityInitializer);
    }

    private void CustomSecurityInitializer(Security security)
    {
        // Overwrite some of the reality models
        security.SetFeeModel(new ConstantFeeModel(1));
    }
}
```

Portfolio Level Models

A portfolio is a semi-closed system where the state updates over time as the value and quantity of assets in the portfolio fluctuate. It's a semi-closed system because the portfolio usually incurs transaction fees and you can add and deposit capital from the portfolio. LEAN models the portfolio holdings to track and update the average price of each asset with every transaction.

You can set the following portfolio-level models:

- Margin call model

  ```C#
  Portfolio.SetMarginCallModel(new DefaultMarginCallModel(Portfolio, DefaultOrderProperties));
  ```

- Risk free interest rate model

  ```C#
  SetRiskFreeInterestRateModel(new InterestRateProvider());
  ```

Reality Modeling

Trade Fills

Trade Fills

Key Concepts

Introduction

A trade fill is the quantity and price at which your brokerage executes your order in the market. Fill models model how each type of order fills to accurately simulate the behavior of a real brokerage. Fill models determine the price and quantity of your fills, can incorporate spread costs, and work with the slippage model to add slippage into the fill price. If you trade US Equities, our built-in fill models can fill your orders at the official opening and closing auction prices .

Set Models

The brokerage model of your algorithm automatically sets the fill model for each security, but you can override it. To manually set the fill model of a security, call the `SetFillModel` method on the Security object.

C#

```
public override void Initialize()
{
    var security = AddEquity("SPY");
    // Set the fill model for the requested security to backtest with the most realistic scenario
    // ImmediateFillModel provide no delay from network issue or brokerage rules
    security.SetFillModel(new ImmediateFillModel());
}
```

You can also set the fill model in a security initializer . If your algorithm has a dynamic universe, use the security initializer technique. In order to initialize single security subscriptions with the security initializer, call `AddSecurityInitializer` before you create the subscriptions.

C#

```
public class AddSecurityInitializerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the security initializer to set models of assets.
        AddSecurityInitializer(CustomSecurityInitializer);
    }

    private void CustomSecurityInitializer(Security security)
    {
        // Overwrite some of the reality models
        security.SetFillModel(new ImmediateFillModel());
    }
}
```

To view all the pre-built fill models, see Supported Models .

Default Behavior

The brokerage model of your algorithm automatically sets the fill model of each security. The default brokerage model is the `DefaultBrokerageModel` , which sets the EquityFillModel for Equities, the FutureFillModel for Futures, the FutureOptionFillModel for Future Options, and the ImmediateFillModel for all other asset classes.

Model Structure

Fill Models should extend the `FillModel` class. To implement your own fill model, override the methods in the `FillModel` class you wish to change. The class has a dedicated method for each order type. Most of the methods receive a `Security` and `Order` object and return an `OrderEvent` object that contains information about the order status, fill quantity, and errors.

C#

```
public class CustomFillModelExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
     SetStartDate(2024, 9, 1);
     SetEndDate(2024, 12, 31);
        // In the Initialize method, set the custom fill model for an added security to use the custom model
        var security = AddEquity("SPY");
        security.SetFillModel(new MyFillModel());
    }
}

// Define the custom fill model outside of the algorithm
public class MyFillModel : FillModel {

    public override OrderEvent MarketFill(Security asset, MarketOrder order) {
        return base.MarketFill(asset, order);
    }

    public override OrderEvent LimitFill(Security asset, LimitOrder order) {
        return base.LimitFill(asset, order);
    }

    public override OrderEvent LimitIfTouchedFill(Security asset, LimitIfTouchOrder order) {
        return base.LimitIfTouchedFill(asset, order);
    }

    public override OrderEvent StopMarketFill(Security asset, StopMarketOrder order) {
        return base.StopMarketFill(asset, order);
    }

    public override OrderEvent StopLimitFill(Security asset, StopLimitOrder order) {
        return base.StopLimitFill(asset, order);
    }

    public override OrderEvent TrailingStopFill(Security asset, TrailingStopOrder order) {
     return TrailingStopFill(asset, order);
    }

    public override OrderEvent MarketOnOpenFill(Security asset, MarketOnOpenOrder order) {
        return base.MarketOnOpenFill(asset, order);
    }

    public override OrderEvent MarketOnCloseFill(Security asset, MarketOnCloseOrder order) {
        return base.MarketOnCloseFill(asset, order);
    }

    public override List<OrderEvent> ComboMarketFill(Order order, FillModelParameters parameters) {
        return base.ComboMarketFill(order, parameters);
    }

    public override List<OrderEvent> ComboLimitFill(Order order, FillModelParameters parameters) {
        return base.ComboLimitFill(order, parameters);
    }

    public override List<OrderEvent> ComboLegLimitFill(Order order, FillModelParameters parameters) {
        return base.ComboLegLimitFill(order, parameters);
    }
}
```

The `FillModelParameters` class has the following properties:

Partial Fills

In live trading, your orders can partially fill. For example, if you have a buy limit order at the bid price for 100 shares and someone sells 10 shares with a market order, your order is partially filled. In backtests, the pre-built fill models assume orders completely fill. To simulate partial fills in backtests, create a custom fill model.

Stale Fills

Stale fills occur when you fill an order with price data that is timestamped an hour or more into the past. Stale fills usually only occur if you trade illiquid assets or if your algorithm uses daily data but you trade intraday with Scheduled Events. If your order is filled with stale data, the fill price may not be realistic. The pre-built fill models can only fill market orders with stale data. To adjust the length of time that needs to pass before an order is considered stale, set the `StalePriceTimeSpan` setting.

```C#
public override void Initialize()
{
    // Adjust the stale price time span to be 10 minutes.
    Settings.StalePriceTimeSpan = TimeSpan.FromMinutes(10);
}
```

Examples

The following examples demonstrate common practices for implementing a custom fill model.

Example 1: Default Model Wrapper

The following algorithm defines a subclass of the default `FillModel` and overrides its built-in members so you can easily customize the parts you need.

C#

```
public class CustomFillModelAlgorithm : QCAlgorithm
{

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 10, 15);
        SetCash(100000);

        var security = AddEquity("SPY", Resolution.Daily);
        security.SetFillModel(new MyFillModel());
    }

    public override void OnData(Slice data)
    {
        if (!Portfolio.Invested)
        {
            SetHoldings("SPY", 1);
        }
    }
}

public class MyFillModel : FillModel {

    public override OrderEvent MarketFill(Security asset, MarketOrder order) {
        return base.MarketFill(asset, order);
    }

    public override OrderEvent LimitFill(Security asset, LimitOrder order) {
        return base.LimitFill(asset, order);
    }

    public override OrderEvent LimitIfTouchedFill(Security asset, LimitIfTouchedOrder order) {
        return base.LimitIfTouchedFill(asset, order);
    }

    public override OrderEvent StopMarketFill(Security asset, StopMarketOrder order) {
        return base.StopMarketFill(asset, order);
    }

    public override OrderEvent StopLimitFill(Security asset, StopLimitOrder order) {
        return base.StopLimitFill(asset, order);
    }

    public override OrderEvent TrailingStopFill(Security asset, TrailingStopOrder order) {
        return TrailingStopFill(asset, order);
    }

    public override OrderEvent MarketOnOpenFill(Security asset, MarketOnOpenOrder order) {
        return base.MarketOnOpenFill(asset, order);
    }

    public override OrderEvent MarketOnCloseFill(Security asset, MarketOnCloseOrder order) {
        return base.MarketOnCloseFill(asset, order);
    }

    public override List<OrderEvent> ComboMarketFill(Order order, FillModelParameters parameters) {
        return base.ComboMarketFill(order, parameters);
    }

    public override List<OrderEvent> ComboLimitFill(Order order, FillModelParameters parameters) {
        return base.ComboLimitFill(order, parameters);
    }

    public override List<OrderEvent> ComboLegLimitFill(Order order, FillModelParameters parameters) {
        return base.ComboLegLimitFill(order, parameters);
    }
}
```

Example 2: Volume Share Fill

The following algorithm longs the top 10 liquid constituents and short the bottom 10 liquid constituents of QQQ. To realistically fill the less liquid stocks, we implement a fill model that only fills with at most 30% of the previous second bar.

C#

```
public class VolumeShareSlippageModelAlgorithm : QCAlgorithm
{
    private List<Symbol> _longs = new();
    private List<Symbol> _shorts = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // The security initializer applies the VolumeShareFillModel to all assets.
        Settings.SeedInitialPrices = true;
        AddSecurityInitializer(security =>
        {
            // Create a slippage model to fill only 30% of the volume of the previous second bar to fill illiquid
stocks realistically.
            security.SetFillModel(new VolumeShareFillModel(this, 0.3m));
        });

        // Request extended market hour SPY data for trading.
        var qqq = AddEquity("QQQ").Symbol;

        // Weekly portfolio updating to allow time to capitalize on the popularity gap.
        UniverseSettings.Schedule.On(DateRules.WeekStart());
        // Add universe to trade on the most and least liquid stocks among QQQ constituents.
        AddUniverse(
            // First, we select all QQQ constituents for the next filter on liquidity.
            Universe.ETF(qqq, Market.USA, UniverseSettings, (constituents) => constituents.Select(c => c.Symbol)),
            FundamentalSelection
        );

        // Set a scheduled event to rebalance the portfolio at the start of every week.
        Schedule.On(
            DateRules.WeekStart(qqq),
            TimeRules.AfterMarketOpen(qqq),
            Rebalance
        );
    }

    private IEnumerable<Symbol> FundamentalSelection(IEnumerable<Fundamental> fundamentals)
    {
        var sortedByDollarVolume = fundamentals.OrderBy(x => x.DollarVolume).ToList();
        // Add the 10 most liquid stocks to the universe to long later.
        _longs = sortedByDollarVolume.TakeLast(10)
            .Select(x => x.Symbol)
            .ToList();
        // Add the 10 least liquid stocks to the universe to short later.
        _shorts = sortedByDollarVolume.Take(10)
            .Select(x => x.Symbol)
            .ToList();

        return _longs.Union(_shorts);
    }

    private void Rebalance()
    {
        // Equally invest in the selected stocks to dissipate capital risk evenly.
        // Dollar neutral of long and short stocks to eliminate systematic risk, only capitalize the popularity gap.
        var targets = _longs.Select(symbol => new PortfolioTarget(symbol, 0.05m)).ToList();
        targets.AddRange(_shorts.Select(symbol => new PortfolioTarget(symbol, -0.05m)).ToList());

        // Liquidate the stocks that are not the most and least popular to release funds for higher expected return
trades.
        SetHoldings(targets, liquidateExistingHoldings: true);
    }

    private class VolumeShareFillModel : FillModel
    {
        private readonly QCAlgorithm _algorithm;
        private readonly decimal _maximumRatio;
        private readonly Dictionary<int, decimal> _absoluteRemainingByOrderId = new();

        public VolumeShareFillModel(QCAlgorithm algorithm, decimal maximumRatio = 0.3m)
            : base()
        {
            _algorithm = algorithm;
            _maximumRatio = maximumRatio;
        }

        public override OrderEvent MarketFill(Security asset, MarketOrder order)
        {
            decimal absoluteRemaining;
```

```
            decimal absoluteRemaining;
            if (!_absoluteRemainingByOrderId.TryGetValue(order.Id, out absoluteRemaining))
            {
                absoluteRemaining = order.AbsoluteQuantity;
            }

            // Create the object
            var fill = base.MarketFill(asset, order);

            // Set the fill amount
            fill.FillQuantity = Math.Sign(order.Quantity) * 10m;
            if (Math.Min(Math.Abs(fill.FillQuantity), absoluteRemaining) == absoluteRemaining)
            {
                fill.FillQuantity = Math.Sign(order.Quantity) * absoluteRemaining;
                fill.Status = OrderStatus.Filled;
                _absoluteRemainingByOrderId.Remove(order.Id);
            }
            else
            {
                fill.Status = OrderStatus.PartiallyFilled;
                _absoluteRemainingByOrderId[order.Id] = absoluteRemaining - Math.Abs(fill.FillQuantity);
                var price = fill.FillPrice;
                //_algorithm.Debug($"{_algorithm.Time} - Partial Fill - Remaining
{_absoluteRemainingByOrderId[order.Id]} Price - {price}");
            }
            return fill;
        }
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
CustomModelsAlgorithm.cs C# CustomPartialFillModelAlgorithm.cs C# ComboOrdersFillModelAlgorithm.cs C#
ForwardDataOnlyFillModelAlgorithm.cs C#

Trade Fills

Supported Models

Fill models determine the price and quantity of your fills. The following fill models are available:

Equity Model

Equity

Future Model

Futures

Future Option Model

Future Options

Immediate Model

All asset classes

Latest Price Model

All asset classes

See Also

Reality Modeling
Order Types

Supported Models

Equity Model

Introduction

The `EquityFillModel` is the default fill model if you trade Equity assets with the DefaultBrokerageModel . This fill model fills trades completely and immediately.

```
C#
security.SetFillModel(new EquityFillModel());
```

The fill logic of each order depends on the order type, the data format of the security subscription, and the order direction. The following sections explain the fill logic of each order given these factors.

For more information about this model, see the class reference and implementation .

Market Orders

The model fills buy market orders at the best effort ask price plus slippage and fills sell market orders at the best effort bid price minus slippage.

To get the best effort bid price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a buy quote, use the bid price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing bid price of the most recent `QuoteBar` .

To get the best effort ask price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a sell quote, use the ask price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing ask price of the most recent `QuoteBar` .

If neither of the preceding procedures yield a result, the model uses the following procedure to get the best effort bid or ask price:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a tick of type `TickType.Trade` , use the last trade price.
2. If the subscription provides `TradeBar` data, use the closing bid price of the most recent `QuoteBar` .

The model only fills market orders during regular trading hours.

Limit Orders

To fill limit orders, the model first gets the best effort `TradeBar` . To get the best effort `TradeBar` , the model checks the resolution of your security subscription. If your subscription provies `Tick` data, it gets the most recent batch of trade ticks and consolidates them to build a `TradeBar` . If your subscription provies `TradeBar` data, it gets the most recent `TradeBar` . If the `EndTime` of the best effort `TradeBar` is less than or equal to the time you placed the order, the model waits until the next best effort `TradeBar` to fill the order.

Once the model has a valid best effort `TradeBar` , it can fill the order. The following table shows the fill condition and fill price of limit orders. The model only fills the order once the fill condition is met.

Order Direction	Fill Condition	Fill Price
Buy	low price < limit price	min(open price, limit price)
Sell	high price > limit price	max(open price, limit price)

The model only fills limit orders when the exchange is open.

The model won't fill limit orders with stale data or data with the order timestamp to avoid look-ahead bias.

Limit if Touched Orders

To fill limit if touched orders, the model first gets the best effort `TradeBar` . To get the best effort `TradeBar` , the model checks the resolution of your security subscription. If your subscription provies `Tick` data, it gets the most recent batch of trade ticks and consolidates them to build a `TradeBar` . If your subscription provies `TradeBar` data, it gets the most recent `TradeBar` . If the `EndTime` of the best effort `TradeBar` is less than or equal to the time you placed the order, the model waits until the next best effort `TradeBar` to fill the order.

After the model has a valid best effort `TradeBar` , it can check if the trigger price has been touched. The following table describes the trigger condition of limit if touched orders for each order direction:

Order Direction	Trigger Condition
Buy	low price <= trigger price
Sell	high price >= trigger price

Once the limit if touched order triggers, the model starts to check if it should fill the order. The following table shows the fill condition and fill price of limit if touched orders. The model only fills the order once the fill condition is met

Order Direction	Fill Condition	Fill Price
Buy	Best effort ask price <= limit price	min(best effort ask price, limit price)
Sell	Best effort bid price >= limit price	max(best effort bid price, limit price)

To get the best effort bid price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a buy quote, use the bid price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing bid price of the most recent `QuoteBar` .

To get the best effort ask price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a sell quote, use the ask price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing ask price of the most recent `QuoteBar` .

If neither of the preceding procedures yield a result, the model uses the following procedure to get the best effort bid or ask price:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a tick of type `TickType.Trade` , use the last trade price.
2. If the subscription provides `TradeBar` data, use the closing bid price of the most recent `QuoteBar` .

The model only fills limit orders when the exchange is open.

The model won't trigger or fill limit if touched orders with stale data .

Stop Market Orders

To fill stop market orders, the model first gets the best effort `TradeBar` . To get the best effort `TradeBar` , the model checks the resolution of your security subscription. If your subscription provies `Tick` data, it gets the most recent batch of trade ticks and consolidates them to build a `TradeBar` . If your subscription provies `TradeBar` data, it gets the most recent `TradeBar` . If the `EndTime` of the best effort `TradeBar` is less than or equal to the time you placed the order, the model waits until the next best effort `TradeBar` to fill the order.

Once the stop condition is met, the model fills the orders and sets the fill price.

Once the model has a valid best effort `TradeBar` , it can fill the order. The following table shows the stop condition and fill price of stop market orders. The model only fills the order once the stop condition is met.

Order Direction	Fill Condition	Fill Price
Buy	high price >= stop price	max(open price, stop price) + slippage
Sell	low price <= stop price	min(open price, stop price) - slippage

The model only fills stop market orders during regular trading hours.

The model won't fill stop market orders with stale data or data with the order timestamp to avoid look-ahead bias.

Stop Limit Orders

The fill logic of stop limit orders depends on the data format of the security subscription and the order direction. The following table shows the fill price of stop limit orders given these factors. To determine the fill price of the order, the fill model first checks the most recent tick for the security. If your security subscription doesn't provide tick data, the fill model checks the most recent `QuoteBar` . If your security subscription doesn't provide quote data, the fill model checks the most recent `TradeBar` .

The following table describes how the fill model processes the order given the data format and order direction. Once the stop condition is met, the model starts to check the fill condition. Once the fill condition is met, the model fills the orders and sets the fill price.

Data Format	TickType	Order Direction	Stop Condition	Fill Condition	Fill Price
Tick	Quote	Buy	quote price > stop price	quote price < limit price	min(quote price, limit price)
Tick	Quote	Sell	quote price < stop price	quote price > limit price	max(quote price, limit price)
Tick	Trade	Buy	trade price > stop price	trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	trade price < stop price	trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	ask high price > stop price	ask close price < limit price	min(ask high price, limit price)
QuoteBar		Sell	bid low price < stop price	bid close price > limit price	max(bid low price, limit price)
TradeBar		Buy	high price > stop price	close price < limit price	min(high price, limit price)
TradeBar		Sell	low price < stop price	close price > limit price	max(low price, limit price)

The model won't fill stop limit orders with stale data or data with the order timestamp to avoid look-ahead bias.

The model only fills stop limit orders when the exchange is open.

Trailing Stop Orders

To fill trailing stop orders, the model first gets the best effort `TradeBar` . To get the best effort `TradeBar` , the model checks the resolution of your security subscription. If your subscription provies `Tick` data, it gets the most recent batch of trade ticks and consolidates them to build a `TradeBar` . If your subscription provies `TradeBar` data, it gets the most recent `TradeBar` . If the `EndTime` of the best effort `TradeBar` is less than or equal to the time you placed the order, the model waits until the next best effort `TradeBar` to fill the order.

Once the stop condition is met, the model fills the orders and sets the fill price.

Once the model has a valid best effort `TradeBar` , it can fill the order. The following table shows the stop condition and fill price of trailing stop orders. The model only fills the order once the stop condition is met.

Order Direction	Fill Condition	Fill Price
Buy	high price >= stop price	max(open price, stop price) + slippage
Sell	low price <= stop price	min(open price, stop price) - slippage

While the stop condition is not met, the model updates the stop price under certain conditions. The following table shows the update condition and stop price value for currency-based trailing amounts:

Order Direction	Update Condition	Stop Price
Buy	stop price - low price <= trailing amount	low price + trailing amount
Sell	high price - stop price <= trailing amount	high price - trailing amount

The following table shows the update condition and stop price value for percentage-based trailing amounts:

Order Direction	Update Condition	Stop Price
Buy	stop price - low price <= low price * trailing amount	low price * (1 + trailing amount)
Sell	high price - stop price <= high price * trailing amount	high price * (1 - trailing amount)

The model only fills trailing stop orders during regular trading hours.

The model won't fill trailing stop orders with stale data or data with the order timestamp to avoid look-ahead bias.

Market on Open Orders

The following table describes the fill price of market on open orders for each data format and order direction:

Data Format	Order Direction	Fill Price
`Tick`	Buy	If the model receives the official opening auction price within one minute, the order fills at official open price + slippage. After one minute, the order fills at the most recent trade price + slippage. If the security doesn't trade within the first two minutes, the order fills at the best effort ask price + slippage.
`Tick`	Sell	If the model receives the official opening auction price within one minute, the order fills at the official open price - slippage. After one minute, the order fills at the most recent trade price - slippage. If the security doesn't trade within the first two minutes, the order fills at the best effort bid price - slippage.
`TradeBar`	Buy	Open price + slippage
`TradeBar`	Sell	Open price - slippage
`QuoteBar`	Buy	Best effort ask price + slippage
`QuoteBar`	Sell	Best effort bid price - slippage

The model checks the data format in the following order:

1. `Tick`

2. `TradeBar`
3. `QuoteBar`

To get the best effort bid price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a buy quote, use the bid price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing bid price of the most recent `QuoteBar` .

To get the best effort ask price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a sell quote, use the ask price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing ask price of the most recent `QuoteBar` .

If neither of the preceding procedures yield a result, the model uses the following procedure to get the best effort bid or ask price:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a tick of type `TickType.Trade` , use the last trade price.
2. If the subscription provides `TradeBar` data, use the closing bid price of the most recent `QuoteBar` .

Market on Close Orders

The following table describes the fill price of market on close orders for each data format and order direction:

Data Format	Order Direction	Fill Price
`Tick`	Buy	If the model receives the official closing auction price within one minute after the close, the order fills at official close price + slippage. After one minute, the order fills at the most recent trade price + slippage. If the security doesn't trade within the first two minutes, the order fills at the best effort ask price + slippage.
`Tick`	Sell	If the model receives the official closing auction price within one minute after the close, the order fills at the official close price - slippage. After one minute, the order fills at the most recent trade price - slippage. If the security doesn't trade within the first two minutes after the close, the order fills at the best effort bid price - slippage.
`TradeBar`	Buy	Open price + slippage
`TradeBar`	Sell	Open price - slippage
`QuoteBar`	Buy	Best effort ask price + slippage
`QuoteBar`	Sell	Best effort bid price - slippage

The model checks the data format in the following order:

1. `Tick`
2. `TradeBar`
3. `QuoteBar`

To get the best effort bid price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a buy quote, use the bid price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing bid price of the most recent `QuoteBar` .

To get the best effort ask price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a sell quote, use the ask price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing ask price of the most recent `QuoteBar` .

If neither of the preceding procedures yield a result, the model uses the following procedure to get the best effort bid or ask price:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a tick of type `TickType.Trade` , use the last trade price.
2. If the subscription provides `TradeBar` data, use the closing bid price of the most recent `QuoteBar` .

Combo Market Orders

The fill logic of combo market orders depends on the data format of the security subscription and the order direction. The following table shows the fill price of combo market orders given these factors. To determine the fill price of the order, the fill model first checks the most recent tick for the security. If your security subscription doesn't provide tick data, the fill model checks the most recent `QuoteBar` . If your security subscription doesn't provide quote data, the fill model checks the most recent `TradeBar` .

Data Format	`TickType`	Order Direction	Fill Price
`Tick`	`Quote`	Buy	Ask quote price + slippage
`Tick`	`Quote`	Sell	Bid quote price - slippage
`Tick`	`Trade`	Buy	Trade price + slippage
`Tick`	`Trade`	Sell	Trade price - slippage
`QuoteBar`		Buy	Ask close price + slippage
`QuoteBar`		Sell	Bid close price - slippage
`TradeBar`		Buy	Close price + slippage
`TradeBar`		Sell	Close price - slippage

The model only fills combo market orders if all the following conditions are met:

- The exchange is open
- The data isn't stale

- All the legs can fill in the same time step after the order time step

The fill quantity of each leg is the product of the leg order quantity and the combo market order quantity.

Combo Limit Orders

The fill logic of combo limit orders depends on the data format of the security subscription and the order direction. To determine the fill price of the order, the fill model first checks the most recent tick for the security. If your security subscription doesn't provide tick data, the fill model checks the most recent `QuoteBar` . If your security subscription doesn't provide quote data, the fill model checks the most recent `TradeBar` .

To fill combo limit orders, the fill model calculates the aggregate price of the combo order, which is the sum of prices for each security in the order legs. The price of each security is a function of the data format and order direction. Legs with a positive order quantity increase the aggregate price and legs with a negative quantity decrease the aggregate price. The following table shows how the fill model calculates the security prices.

Data Format	`TickType`	Combo Order Direction	Leg Order Direction	Price
`Tick`	`Quote`	Buy or sell	Buy	Ask price
`Tick`	`Quote`	Buy or sell	Sell	Bid price
`Tick`	`Trade`	Buy or sell	Buy or sell	Trade price
`QuoteBar`		Buy	Buy	Ask low price
`QuoteBar`		Buy	Sell	Bid low price
`QuoteBar`		Sell	Buy	Ask high price
`QuoteBar`		Sell	Sell	Bid high price
`TradeBar`		Buy	Buy or sell	Low price
`TradeBar`		Sell	Buy or sell	High price

After the fill model calculates the aggregate price of the combo order, it checks if it should fill the order. The following table describes the fill condition of the combo order and the fill price price of each leg:

Data Format	TickType	Combo Order Direction	Fill Condition	Leg Order Direction	Fill Price
Tick	Quote	Buy	Aggregate price < combo limit price	Buy or sell	Quote price
Tick	Quote	Sell	Aggregate price > combo limit price	Buy or sell	Quote price
Tick	Trade	Buy	Aggregate price < combo limit price	Buy or sell	Trade price
Tick	Trade	Sell	Aggregate price > combo limit price	Buy or sell	Trade price
QuoteBar		Buy	Aggregate price < combo limit price	Buy	Ask low price
QuoteBar		Buy	Aggregate price < combo limit price	Sell	Bid low price
QuoteBar		Sell	Aggregate price > combo limit price	Buy	Ask high price
QuoteBar		Sell	Aggregate price > combo limit price	Sell	Bid high price
TradeBar		Buy	Aggregate price < combo limit price	Buy or sell	Low price
TradeBar		Sell	Aggregate price > combo limit price	Buy or sell	High price

The model only fills combo limit orders if the data isn't stale and all the legs can fill in the same time step after the order time step. The fill quantity of each leg is the product of the leg order quantity and the combo order quantity.

Combo Leg Limit Orders

The fill logic of combo leg limit orders depends on the data format of the security subscription and the order direction. The following table shows the fill price of combo leg limit orders given these factors. To determine the fill price of the order, the fill model first checks the most recent tick for the security. If your security subscription doesn't provide tick data, the fill model checks the most recent `QuoteBar` . If your security subscription doesn't provide quote data, the fill model checks the most recent `TradeBar` .

The order direction in the table represents the order direction of the order leg, not the order direction of the combo order.

Data Format	TickType	Order Direction	Fill Condition	Fill Price
Tick	Quote	Buy	Ask price < limit price	min(ask price, limit price)
Tick	Quote	Sell	Bid price > limit price	max(bid price, limit price)
Tick	Trade	Buy	Trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	Trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	Ask low price < limit price	min(ask high price, limit price)
QuoteBar		Sell	Bid high price > limit price	max(bid low price, limit price)
TradeBar		Buy	Low price < limit price	min(high price, limit price)
TradeBar		Sell	High price > limit price	max(low price, limit price)

The model only fills combo leg limit orders if all the following conditions are met:

- The exchange is open
- The data isn't stale
- All the legs can fill in the same time step after the order time step

The fill quantity is the product of the leg order quantity and the combo order quantity.

Supported Models

Future Model

Introduction

The `FutureFillModel` is the default fill model if you trade Futures contracts with the DefaultBrokerageModel . This fill model fills trades completely and immediately.

```
C#
security.SetFillModel(new FutureFillModel());
```

The fill logic of each order depends on the order type, the data format of the security subscription, and the order direction. The following tables show the fill price of each order given these factors. To determine the fill price of an order, the fill model first checks the most recent tick for the security. If your security subscription doesn't provide tick data, the fill model checks the most recent `QuoteBar` . If your security subscription doesn't provide quote data, the fill model checks the most recent `TradeBar` .

For more information about this model, see the class reference and implementation .

Market Orders

The following table describes the fill price of market orders for each data format and order direction:

Data Format	`TickType`	Order Direction	Fill Price
`Tick`	`Quote`	Buy	quote price + slippage
`Tick`	`Quote`	Sell	quote price - slippage
`Tick`	`Trade`	Buy	trade price + slippage
`Tick`	`Trade`	Sell	trade price - slippage
`QuoteBar`		Buy	ask close price + slippage
`QuoteBar`		Sell	bid close price - slippage
`TradeBar`		Buy	close price + slippage
`TradeBar`		Sell	close price - slippage

The model only fills market orders if the exchange is open and it immediately fills them. If your algorithm places a market order at 10 AM, it fills at 10 AM.

Limit Orders

The following table describes the fill logic of limit orders for each data format and order direction:

Data Format	TickType	Order Direction	Fill Condition	Fill Price
Tick	Quote	Buy	quote price < limit price	min(quote price, limit price)
Tick	Quote	Sell	quote price > limit price	max(quote price, limit price)
Tick	Trade	Buy	trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	ask low price < limit price	min(ask high price, limit price)
QuoteBar		Sell	bid high price > limit price	max(bid low price, limit price)
TradeBar		Buy	low price < limit price	min(high price, limit price)
TradeBar		Sell	high price > limit price	max(low price, limit price)

The model won't fill limit orders with stale data or data with the order timestamp to avoid look-ahead bias.

Limit if Touched Orders

The model converts a limit if touched order to a limit order when the trigger condition is met. The following table describes the trigger condition of limit if touched orders for each data format and order direction:

Data Format	TickType	Order Direction	Trigger Condition
Tick	Quote	Buy	quote price <= trigger price
Tick	Quote	Sell	quote price >= trigger price
Tick	Trade	Buy	trade price <= trigger price
Tick	Trade	Sell	trade price >= trigger price
TradeBar		Buy	low price <= trigger price
TradeBar		Sell	high price >= trigger price

Once the limit if touched order triggers, to fill the order, the model checks the bid price for buy orders and the ask price for sell orders.

To get the bid price, the model uses the following procedure:

1. If the subscription provides Tick data and the most recent batch of ticks contains a buy quote, use the bid price of the most recent quote tick.
2. If the subscription provides QuoteBar data, use the closing bid price of the most recent QuoteBar .

To get the ask price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a sell quote, use the ask price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing ask price of the most recent `QuoteBar` .

If neither of the preceding procedures yield a result, the model uses the following procedure to get the bid or ask price:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a tick of type `TickType.Trade` , use the last trade price.
2. If the subscription provides `TradeBar` data, use the closing bid price of the most recent `QuoteBar` .

Buy orders fill when the bid price <= limit price and sell orders fill when the ask price >= limit price. The order fills at the limit price. The model won't trigger or fill limit if touched orders with stale data or data with the order timestamp to avoid look-ahead bias.

Stop Market Orders

The following table describes the fill logic of stop market orders for each data format and order direction. Once the stop condition is met, the model fills the orders and sets the fill price.

Data Format	`TickType`	Order Direction	Stop Condition	Fill Price
`Tick`	`Quote`	Buy	quote price > stop price	max(stop price, quote price + slippage)
`Tick`	`Quote`	Sell	quote price < stop price	min(stop price, quote price - slippage)
`Tick`	`Trade`	Buy	trade price > stop price	max(stop price, last trade price + slippage)
`Tick`	`Trade`	Sell	trade price < stop price	min(stop price, last trade price - slippage)
`QuoteBar`		Buy	ask high price > stop price	max(stop price, ask close price + slippage)
`QuoteBar`		Sell	bid low price < stop price	min(stop price, bid close price - slippage)
`TradeBar`		Buy	high price > stop price	max(stop price, close price + slippage)
`TradeBar`		Sell	low price < stop price	min(stop price, close price - slippage)

The model only fills stop market orders when the exchange is open.

The model won't fill stop market orders with stale data or data with the order timestamp to avoid look-ahead bias.

Stop Limit Orders

The following table describes the fill logic of stop limit orders for each data format and order direction. Once the stop condition

is met, the model starts to check the fill condition. Once the fill condition is met, the model fills the orders and sets the fill price.

Data Format	TickType	Order Direction	Stop Condition	Fill Condition	Fill Price
Tick	Quote	Buy	quote price > stop price	quote price < limit price	min(quote price, limit price)
Tick	Quote	Sell	quote price < stop price	quote price > limit price	max(quote price, limit price)
Tick	Trade	Buy	trade price > stop price	trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	trade price < stop price	trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	ask high price > stop price	ask close price < limit price	min(ask high price, limit price)
QuoteBar		Sell	bid low price < stop price	bid close price > limit price	max(bid low price, limit price)
TradeBar		Buy	high price > stop price	close price < limit price	min(high price, limit price)
TradeBar		Sell	low price < stop price	close price > limit price	max(low price, limit price)

The model won't fill stop limit orders with stale data or data with the order timestamp to avoid look-ahead bias.

Trailing Stop Orders

The following table describes the fill logic of trailing stop orders for each data format and order direction. Once the stop condition is met, the model fills the orders and sets the fill price.

Data Format	TickType	Order Direction	Stop Condition	Fill Price
Tick	Quote	Buy	quote price > stop price	max(stop price, quote price + slippage)
Tick	Quote	Sell	quote price < stop price	min(stop price, quote price - slippage)
Tick	Trade	Buy	trade price > stop price	max(stop price, last trade price + slippage)
Tick	Trade	Sell	trade price < stop price	min(stop price, last trade price - slippage)
QuoteBar		Buy	ask high price > stop price	max(stop price, ask close price + slippage)
QuoteBar		Sell	bid low price < stop price	min(stop price, bid close price - slippage)
TradeBar		Buy	high price > stop price	max(stop price, close price + slippage)
TradeBar		Sell	low price < stop price	min(stop price, close price - slippage)

While the stop condition is not met, the model updates the stop price under certain conditions. The following table shows the update condition and stop price value for currency-based trailing amounts:

Data Format	TickType	Order Direction	Update Condition	Stop Price
Tick	Quote	Buy	stop price - quote price <= trailing amount	quote price + trailing amount
Tick	Quote	Sell	quote price - stop price <= trailing amount	quote price - trailing amount
Tick	Trade	Buy	stop price - trade price <= trailing amount	trade price + trailing amount
Tick	Trade	Sell	trade price - stop price <= trailing amount	trade price - trailing amount
QuoteBar		Buy	stop price - ask low price <= trailing amount	ask low price + trailing amount
QuoteBar		Sell	bid high price - stop price <= trailing amount	bid high price - trailing amount
TradeBar		Buy	stop price - low price <= trailing amount	low price + trailing amount
TradeBar		Sell	high price - stop price <= trailing amount	high price - trailing amount

The following table shows the update condition and stop price value for percentage-based trailing amounts.

Data Format	TickType	Order Direction	Update Condition	Stop Price
Tick	Quote	Buy	stop price - quote price <= quote price * trailing amount	quote price * (1 + trailing amount)
Tick	Quote	Sell	quote price - stop price <= quote price * trailing amount	quote price * (1 - trailing amount)
Tick	Trade	Buy	stop price - trade price <= trade price * trailing amount	trade price * (1 + trailing amount)
Tick	Trade	Sell	trade price - stop price <= trade price * trailing amount	trade price * (1 - trailing amount)
QuoteBar		Buy	stop price - ask low price <= ask low price * trailing amount	ask low price * (1 + trailing amount)
QuoteBar		Sell	bid high price - stop price <= bid high price * trailing amount	bid high price * (1 - trailing amount)
TradeBar		Buy	stop price - low price <= stop price * trailing amount	low price * (1 + trailing amount)
TradeBar		Sell	high price - stop price <= high price * trailing amount	high price * (1 - trailing amount)

The model only fills trailing stop orders when the exchange is open.

The model won't fill trailing stop orders with stale data or data with the order timestamp to avoid look-ahead bias.

Market on Open Orders

The following table describes the fill price of market on open orders for each data format and order side:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	quote price + slippage
Tick	Quote	Sell	quote price - slippage
Tick	Trade	Buy	trade price + slippage
Tick	Trade	Sell	trade price - slippage
QuoteBar		Buy	ask open price + slippage
QuoteBar		Sell	bid open price - slippage
TradeBar		Buy	open price + slippage
TradeBar		Sell	open price - slippage

The model won't fill market on open orders during pre-market hours.

The model won't fill market on open orders with stale data or data with the order timestamp to avoid look-ahead bias.

Market on Close Orders

The following table describes the fill price of market on close orders for each data format and order side:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	quote price + slippage
Tick	Quote	Sell	quote price - slippage
Tick	Trade	Buy	trade price + slippage
Tick	Trade	Sell	trade price - slippage
QuoteBar		Buy	ask close price + slippage
QuoteBar		Sell	bid close price - slippage
TradeBar		Buy	close price + slippage
TradeBar		Sell	close price - slippage

The model won't fill market on close orders with stale data or data with the order timestamp to avoid look-ahead bias.

Combo Market Orders

The following table describes the fill price of combo market orders for each data format and order direction:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	Ask quote price + slippage
Tick	Quote	Sell	Bid quote price - slippage
Tick	Trade	Buy	Trade price + slippage
Tick	Trade	Sell	Trade price - slippage
QuoteBar		Buy	Ask close price + slippage
QuoteBar		Sell	Bid close price - slippage
TradeBar		Buy	Close price + slippage
TradeBar		Sell	Close price - slippage

The model only fills combo market orders if all the following conditions are met:

- The exchange is open
- The data isn't stale
- All the legs can fill in the same time step after the order time step

The fill quantity of each leg is the product of the leg order quantity and the combo market order quantity.

Combo Limit Orders

To fill combo limit orders, the fill model calculates the aggregate price of the combo order, which is the sum of prices for each security in the order legs. The price of each security is a function of the data format and order direction. Legs with a positive order quantity increase the aggregate price and legs with a negative quantity decrease the aggregate price. The following table shows how the fill model calculates the security prices.

Data Format	TickType	Combo Order Direction	Leg Order Direction	Price
Tick	Quote	Buy or sell	Buy	Ask price
Tick	Quote	Buy or sell	Sell	Bid price
Tick	Trade	Buy or sell	Buy or sell	Trade price
QuoteBar		Buy	Buy	Ask low price
QuoteBar		Buy	Sell	Bid low price
QuoteBar		Sell	Buy	Ask high price
QuoteBar		Sell	Sell	Bid high price
TradeBar		Buy	Buy or sell	Low price
TradeBar		Sell	Buy or sell	High price

After the fill model calculates the aggregate price of the combo order, it checks if it should fill the order. The following table describes the fill condition of the combo order and the fill price price of each leg:

Data Format	TickType	Combo Order Direction	Fill Condition	Leg Order Direction	Fill Price
Tick	Quote	Buy	Aggregate price < combo limit price	Buy or sell	Quote price
Tick	Quote	Sell	Aggregate price > combo limit price	Buy or sell	Quote price
Tick	Trade	Buy	Aggregate price < combo limit price	Buy or sell	Trade price
Tick	Trade	Sell	Aggregate price > combo limit price	Buy or sell	Trade price
QuoteBar		Buy	Aggregate price < combo limit price	Buy	Ask low price
QuoteBar		Buy	Aggregate price < combo limit price	Sell	Bid low price
QuoteBar		Sell	Aggregate price > combo limit price	Buy	Ask high price
QuoteBar		Sell	Aggregate price > combo limit price	Sell	Bid high price
TradeBar		Buy	Aggregate price < combo limit price	Buy or sell	Low price
TradeBar		Sell	Aggregate price > combo limit price	Buy or sell	High price

The model only fills combo limit orders if the data isn't stale and all the legs can fill in the same time step after the order time step. The fill quantity of each leg is the product of the leg order quantity and the combo order quantity.

Combo Leg Limit Orders

The following table describes the fill logic of combo leg limit orders for each data format and order direction. The order direction in the table represents the order direction of the order leg, not the order direction of the combo order.

Data Format	TickType	Order Direction	Fill Condition	Fill Price
Tick	Quote	Buy	Ask price < limit price	min(ask price, limit price)
Tick	Quote	Sell	Bid price > limit price	max(bid price, limit price)
Tick	Trade	Buy	Trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	Trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	Ask low price < limit price	min(ask high price, limit price)
QuoteBar		Sell	Bid high price > limit price	max(bid low price, limit price)
TradeBar		Buy	Low price < limit price	min(high price, limit price)
TradeBar		Sell	High price > limit price	max(low price, limit price)

The model only fills combo leg limit orders if all the following conditions are met:

- The exchange is open
- The data isn't stale
- All the legs can fill in the same time step after the order time step

The fill quantity is the product of the leg order quantity and the combo order quantity.

Supported Models

Future Option Model

Introduction

The `FutureOptionFillModel` is the default fill model if you trade Future Option contracts with the DefaultBrokerageModel . This fill model fills trades completely and immediately.

```
C#
security.SetFillModel(new FutureOptionFillModel());
```

The fill logic of each order depends on the order type, the data format of the security subscription, and the order direction. The following tables show the fill price of each order given these factors. To determine the fill price of an order, the fill model first checks the most recent tick for the security. If your security subscription doesn't provide tick data, the fill model checks the most recent `QuoteBar` . If your security subscription doesn't provide quote data, the fill model checks the most recent `TradeBar` .

For more information about this model, see the class reference and implementation .

Market Orders

The following table describes the fill price of market orders for each data format and order direction:

Data Format	`TickType`	Order Direction	Fill Price
`Tick`	`Quote`	Buy	quote price + slippage
`Tick`	`Quote`	Sell	quote price - slippage
`Tick`	`Trade`	Buy	trade price + slippage
`Tick`	`Trade`	Sell	trade price - slippage
`QuoteBar`		Buy	ask close price + slippage
`QuoteBar`		Sell	bid close price - slippage
`TradeBar`		Buy	close price + slippage
`TradeBar`		Sell	close price - slippage

The model only fills market orders if the exchange is open and it immediately fills them. If your algorithm places a market order at 10 AM, it fills at 10 AM.

Limit Orders

The following table describes the fill logic of limit orders for each data format and order direction:

Data Format	TickType	Order Direction	Fill Condition	Fill Price
Tick	Quote	Buy	quote price < limit price	min(quote price, limit price)
Tick	Quote	Sell	quote price > limit price	max(quote price, limit price)
Tick	Trade	Buy	trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	ask low price < limit price	min(ask high price, limit price)
QuoteBar		Sell	bid high price > limit price	max(bid low price, limit price)
TradeBar		Buy	low price < limit price	min(high price, limit price)
TradeBar		Sell	high price > limit price	max(low price, limit price)

The model won't fill limit orders with stale data or data with the order timestamp to avoid look-ahead bias.

Limit if Touched Orders

The model converts a limit if touched order to a limit order when the trigger condition is met. The following table describes the trigger condition of limit if touched orders for each data format and order direction:

Data Format	TickType	Order Direction	Trigger Condition
Tick	Quote	Buy	quote price <= trigger price
Tick	Quote	Sell	quote price >= trigger price
Tick	Trade	Buy	trade price <= trigger price
Tick	Trade	Sell	trade price >= trigger price
TradeBar		Buy	low price <= trigger price
TradeBar		Sell	high price >= trigger price

Once the limit if touched order triggers, to fill the order, the model checks the bid price for buy orders and the ask price for sell orders.

To get the bid price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a buy quote, use the bid price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing bid price of the most recent `QuoteBar` .

To get the ask price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a sell quote, use the ask price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing ask price of the most recent `QuoteBar` .

If neither of the preceding procedures yield a result, the model uses the following procedure to get the bid or ask price:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a tick of type `TickType.Trade` , use the last trade price.
2. If the subscription provides `TradeBar` data, use the closing bid price of the most recent `QuoteBar` .

Buy orders fill when the bid price <= limit price and sell orders fill when the ask price >= limit price. The order fills at the limit price. The model won't trigger or fill limit if touched orders with stale data or data with the order timestamp to avoid look-ahead bias.

Stop Market Orders

The following table describes the fill logic of stop market orders for each data format and order direction. Once the stop condition is met, the model fills the orders and sets the fill price.

Data Format	TickType	Order Direction	Stop Condition	Fill Price
Tick	Quote	Buy	quote price > stop price	max(stop price, quote price + slippage)
Tick	Quote	Sell	quote price < stop price	min(stop price, quote price - slippage)
Tick	Trade	Buy	trade price > stop price	max(stop price, last trade price + slippage)
Tick	Trade	Sell	trade price < stop price	min(stop price, last trade price - slippage)
QuoteBar		Buy	ask high price > stop price	max(stop price, ask close price + slippage)
QuoteBar		Sell	bid low price < stop price	min(stop price, bid close price - slippage)
TradeBar		Buy	high price > stop price	max(stop price, close price + slippage)
TradeBar		Sell	low price < stop price	min(stop price, close price - slippage)

The model only fills stop market orders when the exchange is open.

The model won't fill stop market orders with stale data or data with the order timestamp to avoid look-ahead bias.

Stop Limit Orders

The following table describes the fill logic of stop limit orders for each data format and order direction. Once the stop condition

is met, the model starts to check the fill condition. Once the fill condition is met, the model fills the orders and sets the fill price.

Data Format	TickType	Order Direction	Stop Condition	Fill Condition	Fill Price
Tick	Quote	Buy	quote price > stop price	quote price < limit price	min(quote price, limit price)
Tick	Quote	Sell	quote price < stop price	quote price > limit price	max(quote price, limit price)
Tick	Trade	Buy	trade price > stop price	trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	trade price < stop price	trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	ask high price > stop price	ask close price < limit price	min(ask high price, limit price)
QuoteBar		Sell	bid low price < stop price	bid close price > limit price	max(bid low price, limit price)
TradeBar		Buy	high price > stop price	close price < limit price	min(high price, limit price)
TradeBar		Sell	low price < stop price	close price > limit price	max(low price, limit price)

The model won't fill stop limit orders with stale data or data with the order timestamp to avoid look-ahead bias.

Trailing Stop Orders

The following table describes the fill logic of trailing stop orders for each data format and order direction. Once the stop condition is met, the model fills the orders and sets the fill price.

Data Format	TickType	Order Direction	Stop Condition	Fill Price
Tick	Quote	Buy	quote price > stop price	max(stop price, quote price + slippage)
Tick	Quote	Sell	quote price < stop price	min(stop price, quote price - slippage)
Tick	Trade	Buy	trade price > stop price	max(stop price, last trade price + slippage)
Tick	Trade	Sell	trade price < stop price	min(stop price, last trade price - slippage)
QuoteBar		Buy	ask high price > stop price	max(stop price, ask close price + slippage)
QuoteBar		Sell	bid low price < stop price	min(stop price, bid close price - slippage)
TradeBar		Buy	high price > stop price	max(stop price, close price + slippage)
TradeBar		Sell	low price < stop price	min(stop price, close price - slippage)

While the stop condition is not met, the model updates the stop price under certain conditions. The following table shows the update condition and stop price value for currency-based trailing amounts:

Data Format	TickType	Order Direction	Update Condition	Stop Price
Tick	Quote	Buy	stop price - quote price <= trailing amount	quote price + trailing amount
Tick	Quote	Sell	quote price - stop price <= trailing amount	quote price - trailing amount
Tick	Trade	Buy	stop price - trade price <= trailing amount	trade price + trailing amount
Tick	Trade	Sell	trade price - stop price <= trailing amount	trade price - trailing amount
QuoteBar		Buy	stop price - ask low price <= trailing amount	ask low price + trailing amount
QuoteBar		Sell	bid high price - stop price <= trailing amount	bid high price - trailing amount
TradeBar		Buy	stop price - low price <= trailing amount	low price + trailing amount
TradeBar		Sell	high price - stop price <= trailing amount	high price - trailing amount

The following table shows the update condition and stop price value for percentage-based trailing amounts.

Data Format	TickType	Order Direction	Update Condition	Stop Price
Tick	Quote	Buy	stop price - quote price <= quote price * trailing amount	quote price * (1 + trailing amount)
Tick	Quote	Sell	quote price - stop price <= quote price * trailing amount	quote price * (1 - trailing amount)
Tick	Trade	Buy	stop price - trade price <= trade price * trailing amount	trade price * (1 + trailing amount)
Tick	Trade	Sell	trade price - stop price <= trade price * trailing amount	trade price * (1 - trailing amount)
QuoteBar		Buy	stop price - ask low price <= ask low price * trailing amount	ask low price * (1 + trailing amount)
QuoteBar		Sell	bid high price - stop price <= bid high price * trailing amount	bid high price * (1 - trailing amount)
TradeBar		Buy	stop price - low price <= stop price * trailing amount	low price * (1 + trailing amount)
TradeBar		Sell	high price - stop price <= high price * trailing amount	high price * (1 - trailing amount)

The model only fills trailing stop orders when the exchange is open.

The model won't fill trailing stop orders with stale data or data with the order timestamp to avoid look-ahead bias.

Market on Open Orders

The following table describes the fill price of market on open orders for each data format and order side:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	quote price + slippage
Tick	Quote	Sell	quote price - slippage
Tick	Trade	Buy	trade price + slippage
Tick	Trade	Sell	trade price - slippage
QuoteBar		Buy	ask open price + slippage
QuoteBar		Sell	bid open price - slippage
TradeBar		Buy	open price + slippage
TradeBar		Sell	open price - slippage

The model won't fill market on open orders during pre-market hours.

The model won't fill market on open orders with stale data or data with the order timestamp to avoid look-ahead bias.

Market on Close Orders

The following table describes the fill price of market on close orders for each data format and order side:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	quote price + slippage
Tick	Quote	Sell	quote price - slippage
Tick	Trade	Buy	trade price + slippage
Tick	Trade	Sell	trade price - slippage
QuoteBar		Buy	ask close price + slippage
QuoteBar		Sell	bid close price - slippage
TradeBar		Buy	close price + slippage
TradeBar		Sell	close price - slippage

The model won't fill market on close orders with stale data or data with the order timestamp to avoid look-ahead bias.

Combo Market Orders

The following table describes the fill price of combo market orders for each data format and order direction:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	Ask quote price + slippage
Tick	Quote	Sell	Bid quote price - slippage
Tick	Trade	Buy	Trade price + slippage
Tick	Trade	Sell	Trade price - slippage
QuoteBar		Buy	Ask close price + slippage
QuoteBar		Sell	Bid close price - slippage
TradeBar		Buy	Close price + slippage
TradeBar		Sell	Close price - slippage

The model only fills combo market orders if all the following conditions are met:

- The exchange is open
- The data isn't stale
- All the legs can fill in the same time step after the order time step

The fill quantity of each leg is the product of the leg order quantity and the combo market order quantity.

Combo Limit Orders

To fill combo limit orders, the fill model calculates the aggregate price of the combo order, which is the sum of prices for each security in the order legs. The price of each security is a function of the data format and order direction. Legs with a positive order quantity increase the aggregate price and legs with a negative quantity decrease the aggregate price. The following table shows how the fill model calculates the security prices.

Data Format	TickType	Combo Order Direction	Leg Order Direction	Price
Tick	Quote	Buy or sell	Buy	Ask price
Tick	Quote	Buy or sell	Sell	Bid price
Tick	Trade	Buy or sell	Buy or sell	Trade price
QuoteBar		Buy	Buy	Ask low price
QuoteBar		Buy	Sell	Bid low price
QuoteBar		Sell	Buy	Ask high price
QuoteBar		Sell	Sell	Bid high price
TradeBar		Buy	Buy or sell	Low price
TradeBar		Sell	Buy or sell	High price

After the fill model calculates the aggregate price of the combo order, it checks if it should fill the order. The following table describes the fill condition of the combo order and the fill price price of each leg:

Data Format	TickType	Combo Order Direction	Fill Condition	Leg Order Direction	Fill Price
Tick	Quote	Buy	Aggregate price < combo limit price	Buy or sell	Quote price
Tick	Quote	Sell	Aggregate price > combo limit price	Buy or sell	Quote price
Tick	Trade	Buy	Aggregate price < combo limit price	Buy or sell	Trade price
Tick	Trade	Sell	Aggregate price > combo limit price	Buy or sell	Trade price
QuoteBar		Buy	Aggregate price < combo limit price	Buy	Ask low price
QuoteBar		Buy	Aggregate price < combo limit price	Sell	Bid low price
QuoteBar		Sell	Aggregate price > combo limit price	Buy	Ask high price
QuoteBar		Sell	Aggregate price > combo limit price	Sell	Bid high price
TradeBar		Buy	Aggregate price < combo limit price	Buy or sell	Low price
TradeBar		Sell	Aggregate price > combo limit price	Buy or sell	High price

The model only fills combo limit orders if the data isn't stale and all the legs can fill in the same time step after the order time step. The fill quantity of each leg is the product of the leg order quantity and the combo order quantity.

Combo Leg Limit Orders

The following table describes the fill logic of combo leg limit orders for each data format and order direction. The order direction in the table represents the order direction of the order leg, not the order direction of the combo order.

Data Format	TickType	Order Direction	Fill Condition	Fill Price
Tick	Quote	Buy	Ask price < limit price	min(ask price, limit price)
Tick	Quote	Sell	Bid price > limit price	max(bid price, limit price)
Tick	Trade	Buy	Trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	Trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	Ask low price < limit price	min(ask high price, limit price)
QuoteBar		Sell	Bid high price > limit price	max(bid low price, limit price)
TradeBar		Buy	Low price < limit price	min(high price, limit price)
TradeBar		Sell	High price > limit price	max(low price, limit price)

The model only fills combo leg limit orders if all the following conditions are met:

- The exchange is open
- The data isn't stale
- All the legs can fill in the same time step after the order time step

The fill quantity is the product of the leg order quantity and the combo order quantity.

Supported Models

Immediate Model

Introduction

The `ImmediateFillModel` is the default fill model if you trade non-Equity assets with the DefaultBrokerageModel . This fill model fills trades completely and immediately.

```
C#
security.SetFillModel(new ImmediateFillModel());
```

The fill logic of each order depends on the order type, the data format of the security subscription, and the order direction. The following tables show the fill price of each order given these factors. To determine the fill price of an order, the fill model first checks the most recent tick for the security. If your security subscription doesn't provide tick data, the fill model checks the most recent `QuoteBar` . If your security subscription doesn't provide quote data, the fill model checks the most recent `TradeBar` .

For more information about this model, see the class reference and implementation .

Market Orders

The following table describes the fill price of market orders for each data format and order direction:

Data Format	`TickType`	Order Direction	Fill Price
`Tick`	`Quote`	Buy	quote price + slippage
`Tick`	`Quote`	Sell	quote price - slippage
`Tick`	`Trade`	Buy	trade price + slippage
`Tick`	`Trade`	Sell	trade price - slippage
`QuoteBar`		Buy	ask close price + slippage
`QuoteBar`		Sell	bid close price - slippage
`TradeBar`		Buy	close price + slippage
`TradeBar`		Sell	close price - slippage

The model only fills market orders if the exchange is open and it immediately fills them. If your algorithm places a market order at 10 AM, it fills at 10 AM.

Limit Orders

The following table describes the fill logic of limit orders for each data format and order direction:

Data Format	TickType	Order Direction	Fill Condition	Fill Price
Tick	Quote	Buy	quote price < limit price	min(quote price, limit price)
Tick	Quote	Sell	quote price > limit price	max(quote price, limit price)
Tick	Trade	Buy	trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	ask low price < limit price	min(ask high price, limit price)
QuoteBar		Sell	bid high price > limit price	max(bid low price, limit price)
TradeBar		Buy	low price < limit price	min(high price, limit price)
TradeBar		Sell	high price > limit price	max(low price, limit price)

The model won't fill limit orders with stale data or data with the order timestamp to avoid look-ahead bias.

Limit if Touched Orders

The model converts a limit if touched order to a limit order when the trigger condition is met. The following table describes the trigger condition of limit if touched orders for each data format and order direction:

Data Format	TickType	Order Direction	Trigger Condition
Tick	Quote	Buy	quote price <= trigger price
Tick	Quote	Sell	quote price >= trigger price
Tick	Trade	Buy	trade price <= trigger price
Tick	Trade	Sell	trade price >= trigger price
TradeBar		Buy	low price <= trigger price
TradeBar		Sell	high price >= trigger price

Once the limit if touched order triggers, to fill the order, the model checks the bid price for buy orders and the ask price for sell orders.

To get the bid price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a buy quote, use the bid price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing bid price of the most recent `QuoteBar` .

To get the ask price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a sell quote, use the ask price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing ask price of the most recent `QuoteBar` .

If neither of the preceding procedures yield a result, the model uses the following procedure to get the bid or ask price:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a tick of type `TickType.Trade` , use the last trade price.
2. If the subscription provides `TradeBar` data, use the closing bid price of the most recent `QuoteBar` .

Buy orders fill when the bid price <= limit price and sell orders fill when the ask price >= limit price. The order fills at the limit price. The model won't trigger or fill limit if touched orders with stale data or data with the order timestamp to avoid look-ahead bias.

Stop Market Orders

The following table describes the fill logic of stop market orders for each data format and order direction. Once the stop condition is met, the model fills the orders and sets the fill price.

Data Format	`TickType`	Order Direction	Stop Condition	Fill Price
`Tick`	`Quote`	Buy	quote price > stop price	max(stop price, quote price + slippage)
`Tick`	`Quote`	Sell	quote price < stop price	min(stop price, quote price - slippage)
`Tick`	`Trade`	Buy	trade price > stop price	max(stop price, last trade price + slippage)
`Tick`	`Trade`	Sell	trade price < stop price	min(stop price, last trade price - slippage)
`QuoteBar`		Buy	ask high price > stop price	max(stop price, ask close price + slippage)
`QuoteBar`		Sell	bid low price < stop price	min(stop price, bid close price - slippage)
`TradeBar`		Buy	high price > stop price	max(stop price, close price + slippage)
`TradeBar`		Sell	low price < stop price	min(stop price, close price - slippage)

The model only fills stop market orders when the exchange is open.

The model won't fill stop market orders with stale data or data with the order timestamp to avoid look-ahead bias.

Stop Limit Orders

The following table describes the fill logic of stop limit orders for each data format and order direction. Once the stop condition

is met, the model starts to check the fill condition. Once the fill condition is met, the model fills the orders and sets the fill price.

Data Format	TickType	Order Direction	Stop Condition	Fill Condition	Fill Price
Tick	Quote	Buy	quote price > stop price	quote price < limit price	min(quote price, limit price)
Tick	Quote	Sell	quote price < stop price	quote price > limit price	max(quote price, limit price)
Tick	Trade	Buy	trade price > stop price	trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	trade price < stop price	trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	ask high price > stop price	ask close price < limit price	min(ask high price, limit price)
QuoteBar		Sell	bid low price < stop price	bid close price > limit price	max(bid low price, limit price)
TradeBar		Buy	high price > stop price	close price < limit price	min(high price, limit price)
TradeBar		Sell	low price < stop price	close price > limit price	max(low price, limit price)

The model won't fill stop limit orders with stale data or data with the order timestamp to avoid look-ahead bias.

Trailing Stop Orders

The following table describes the fill logic of trailing stop orders for each data format and order direction. Once the stop condition is met, the model fills the orders and sets the fill price.

Data Format	TickType	Order Direction	Stop Condition	Fill Price
Tick	Quote	Buy	quote price > stop price	max(stop price, quote price + slippage)
Tick	Quote	Sell	quote price < stop price	min(stop price, quote price - slippage)
Tick	Trade	Buy	trade price > stop price	max(stop price, last trade price + slippage)
Tick	Trade	Sell	trade price < stop price	min(stop price, last trade price - slippage)
QuoteBar		Buy	ask high price > stop price	max(stop price, ask close price + slippage)
QuoteBar		Sell	bid low price < stop price	min(stop price, bid close price - slippage)
TradeBar		Buy	high price > stop price	max(stop price, close price + slippage)
TradeBar		Sell	low price < stop price	min(stop price, close price - slippage)

While the stop condition is not met, the model updates the stop price under certain conditions. The following table shows the update condition and stop price value for currency-based trailing amounts:

Data Format	TickType	Order Direction	Update Condition	Stop Price
Tick	Quote	Buy	stop price - quote price <= trailing amount	quote price + trailing amount
Tick	Quote	Sell	quote price - stop price <= trailing amount	quote price - trailing amount
Tick	Trade	Buy	stop price - trade price <= trailing amount	trade price + trailing amount
Tick	Trade	Sell	trade price - stop price <= trailing amount	trade price - trailing amount
QuoteBar		Buy	stop price - ask low price <= trailing amount	ask low price + trailing amount
QuoteBar		Sell	bid high price - stop price <= trailing amount	bid high price - trailing amount
TradeBar		Buy	stop price - low price <= trailing amount	low price + trailing amount
TradeBar		Sell	high price - stop price <= trailing amount	high price - trailing amount

The following table shows the update condition and stop price value for percentage-based trailing amounts.

Data Format	TickType	Order Direction	Update Condition	Stop Price
Tick	Quote	Buy	stop price - quote price <= quote price * trailing amount	quote price * (1 + trailing amount)
Tick	Quote	Sell	quote price - stop price <= quote price * trailing amount	quote price * (1 - trailing amount)
Tick	Trade	Buy	stop price - trade price <= trade price * trailing amount	trade price * (1 + trailing amount)
Tick	Trade	Sell	trade price - stop price <= trade price * trailing amount	trade price * (1 - trailing amount)
QuoteBar		Buy	stop price - ask low price <= ask low price * trailing amount	ask low price * (1 + trailing amount)
QuoteBar		Sell	bid high price - stop price <= bid high price * trailing amount	bid high price * (1 - trailing amount)
TradeBar		Buy	stop price - low price <= stop price * trailing amount	low price * (1 + trailing amount)
TradeBar		Sell	high price - stop price <= high price * trailing amount	high price * (1 - trailing amount)

The model only fills trailing stop orders when the exchange is open.

The model won't fill trailing stop orders with stale data or data with the order timestamp to avoid look-ahead bias.

Market on Open Orders

The following table describes the fill price of market on open orders for each data format and order side:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	quote price + slippage
Tick	Quote	Sell	quote price - slippage
Tick	Trade	Buy	trade price + slippage
Tick	Trade	Sell	trade price - slippage
QuoteBar		Buy	ask open price + slippage
QuoteBar		Sell	bid open price - slippage
TradeBar		Buy	open price + slippage
TradeBar		Sell	open price - slippage

The model won't fill market on open orders during pre-market hours.

The model won't fill market on open orders with stale data or data with the order timestamp to avoid look-ahead bias.

Market on Close Orders

The following table describes the fill price of market on close orders for each data format and order side:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	quote price + slippage
Tick	Quote	Sell	quote price - slippage
Tick	Trade	Buy	trade price + slippage
Tick	Trade	Sell	trade price - slippage
QuoteBar		Buy	ask close price + slippage
QuoteBar		Sell	bid close price - slippage
TradeBar		Buy	close price + slippage
TradeBar		Sell	close price - slippage

The model won't fill market on close orders with stale data or data with the order timestamp to avoid look-ahead bias.

Combo Market Orders

The following table describes the fill price of combo market orders for each data format and order direction:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	Ask quote price + slippage
Tick	Quote	Sell	Bid quote price - slippage
Tick	Trade	Buy	Trade price + slippage
Tick	Trade	Sell	Trade price - slippage
QuoteBar		Buy	Ask close price + slippage
QuoteBar		Sell	Bid close price - slippage
TradeBar		Buy	Close price + slippage
TradeBar		Sell	Close price - slippage

The model only fills combo market orders if all the following conditions are met:

- The exchange is open
- The data isn't stale
- All the legs can fill in the same time step after the order time step

The fill quantity of each leg is the product of the leg order quantity and the combo market order quantity.

Combo Limit Orders

To fill combo limit orders, the fill model calculates the aggregate price of the combo order, which is the sum of prices for each security in the order legs. The price of each security is a function of the data format and order direction. Legs with a positive order quantity increase the aggregate price and legs with a negative quantity decrease the aggregate price. The following table shows how the fill model calculates the security prices.

Data Format	TickType	Combo Order Direction	Leg Order Direction	Price
Tick	Quote	Buy or sell	Buy	Ask price
Tick	Quote	Buy or sell	Sell	Bid price
Tick	Trade	Buy or sell	Buy or sell	Trade price
QuoteBar		Buy	Buy	Ask low price
QuoteBar		Buy	Sell	Bid low price
QuoteBar		Sell	Buy	Ask high price
QuoteBar		Sell	Sell	Bid high price
TradeBar		Buy	Buy or sell	Low price
TradeBar		Sell	Buy or sell	High price

After the fill model calculates the aggregate price of the combo order, it checks if it should fill the order. The following table describes the fill condition of the combo order and the fill price price of each leg:

Data Format	TickType	Combo Order Direction	Fill Condition	Leg Order Direction	Fill Price
Tick	Quote	Buy	Aggregate price < combo limit price	Buy or sell	Quote price
Tick	Quote	Sell	Aggregate price > combo limit price	Buy or sell	Quote price
Tick	Trade	Buy	Aggregate price < combo limit price	Buy or sell	Trade price
Tick	Trade	Sell	Aggregate price > combo limit price	Buy or sell	Trade price
QuoteBar		Buy	Aggregate price < combo limit price	Buy	Ask low price
QuoteBar		Buy	Aggregate price < combo limit price	Sell	Bid low price
QuoteBar		Sell	Aggregate price > combo limit price	Buy	Ask high price
QuoteBar		Sell	Aggregate price > combo limit price	Sell	Bid high price
TradeBar		Buy	Aggregate price < combo limit price	Buy or sell	Low price
TradeBar		Sell	Aggregate price > combo limit price	Buy or sell	High price

The model only fills combo limit orders if the data isn't stale and all the legs can fill in the same time step after the order time step. The fill quantity of each leg is the product of the leg order quantity and the combo order quantity.

Combo Leg Limit Orders

The following table describes the fill logic of combo leg limit orders for each data format and order direction. The order direction in the table represents the order direction of the order leg, not the order direction of the combo order.

Data Format	TickType	Order Direction	Fill Condition	Fill Price
Tick	Quote	Buy	Ask price < limit price	min(ask price, limit price)
Tick	Quote	Sell	Bid price > limit price	max(bid price, limit price)
Tick	Trade	Buy	Trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	Trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	Ask low price < limit price	min(ask high price, limit price)
QuoteBar		Sell	Bid high price > limit price	max(bid low price, limit price)
TradeBar		Buy	Low price < limit price	min(high price, limit price)
TradeBar		Sell	High price > limit price	max(low price, limit price)

The model only fills combo leg limit orders if all the following conditions are met:

- The exchange is open
- The data isn't stale
- All the legs can fill in the same time step after the order time step

The fill quantity is the product of the leg order quantity and the combo order quantity.

Supported Models

Latest Price Model

Introduction

The `LatestPriceFillModel` fills trades completely and immediately.

```
security.SetFillModel(new LatestPriceFillModel());
```

The fill logic of each order depends on the order type, the data format of the security subscription, and the order direction. The following tables show the fill price of each order given these factors. To determine the fill price of an order, the fill model first checks the most recent tick for the security. If your security subscription doesn't provide tick data, the fill model checks the most recent `QuoteBar` . If there is no `QuoteBar` available, it uses the OHLC prices from the `Security` object. If there is a `QuoteBar` available, the fill model gets the most recent `TradeBar` . If the `TradeBar` is more recent, it uses the `TradeBar` data. Otherwise, it uses the `QuoteBar` data.

For more information about this model, see the class reference and implementation .

Market Orders

The following table describes the fill price of market orders for each data format and order direction:

Data Format	`TickType`	Order Direction	Fill Price
`Tick`	`Quote`	Buy	quote price + slippage
`Tick`	`Quote`	Sell	quote price - slippage
`Tick`	`Trade`	Buy	trade price + slippage
`Tick`	`Trade`	Sell	trade price - slippage
`QuoteBar`		Buy	ask close price + slippage
`QuoteBar`		Sell	bid close price - slippage
`TradeBar`		Buy	close price + slippage
`TradeBar`		Sell	close price - slippage

The model only fills market orders if the exchange is open and it immediately fills them. If your algorithm places a market order at 10 AM, it fills at 10 AM.

Limit Orders

The following table describes the fill logic of limit orders for each data format and order direction:

Data Format	TickType	Order Direction	Fill Condition	Fill Price
Tick	Quote	Buy	quote price < limit price	min(quote price, limit price)
Tick	Quote	Sell	quote price > limit price	max(quote price, limit price)
Tick	Trade	Buy	trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	ask low price < limit price	min(ask high price, limit price)
QuoteBar		Sell	bid high price > limit price	max(bid low price, limit price)
TradeBar		Buy	low price < limit price	min(high price, limit price)
TradeBar		Sell	high price > limit price	max(low price, limit price)

The model won't fill limit orders with stale data or data with the order timestamp to avoid look-ahead bias.

Limit if Touched Orders

The model converts a limit if touched order to a limit order when the trigger condition is met. The following table describes the trigger condition of limit if touched orders for each data format and order direction:

Data Format	TickType	Order Direction	Trigger Condition
Tick	Quote	Buy	quote price <= trigger price
Tick	Quote	Sell	quote price >= trigger price
Tick	Trade	Buy	trade price <= trigger price
Tick	Trade	Sell	trade price >= trigger price
TradeBar		Buy	low price <= trigger price
TradeBar		Sell	high price >= trigger price

Once the limit if touched order triggers, to fill the order, the model checks the bid price for buy orders and the ask price for sell orders.

To get the bid price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a buy quote, use the bid price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing bid price of the most recent `QuoteBar` .

To get the ask price, the model uses the following procedure:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a sell quote, use the ask price of the most recent quote tick.
2. If the subscription provides `QuoteBar` data, use the closing ask price of the most recent `QuoteBar` .

If neither of the preceding procedures yield a result, the model uses the following procedure to get the bid or ask price:

1. If the subscription provides `Tick` data and the most recent batch of ticks contains a tick of type `TickType.Trade` , use the last trade price.
2. If the subscription provides `TradeBar` data, use the closing bid price of the most recent `QuoteBar` .

Buy orders fill when the bid price <= limit price and sell orders fill when the ask price >= limit price. The order fills at the limit price. The model won't trigger or fill limit if touched orders with stale data or data with the order timestamp to avoid look-ahead bias.

Stop Market Orders

The following table describes the fill logic of stop market orders for each data format and order direction. Once the stop condition is met, the model fills the orders and sets the fill price.

Data Format	`TickType`	Order Direction	Stop Condition	Fill Price
`Tick`	`Quote`	Buy	quote price > stop price	max(stop price, quote price + slippage)
`Tick`	`Quote`	Sell	quote price < stop price	min(stop price, quote price - slippage)
`Tick`	`Trade`	Buy	trade price > stop price	max(stop price, last trade price + slippage)
`Tick`	`Trade`	Sell	trade price < stop price	min(stop price, last trade price - slippage)
`QuoteBar`		Buy	ask high price > stop price	max(stop price, ask close price + slippage)
`QuoteBar`		Sell	bid low price < stop price	min(stop price, bid close price - slippage)
`TradeBar`		Buy	high price > stop price	max(stop price, close price + slippage)
`TradeBar`		Sell	low price < stop price	min(stop price, close price - slippage)

The model only fills stop market orders when the exchange is open.

The model won't fill stop market orders with stale data or data with the order timestamp to avoid look-ahead bias.

Stop Limit Orders

The following table describes the fill logic of stop limit orders for each data format and order direction. Once the stop condition

is met, the model starts to check the fill condition. Once the fill condition is met, the model fills the orders and sets the fill price.

Data Format	TickType	Order Direction	Stop Condition	Fill Condition	Fill Price
Tick	Quote	Buy	quote price > stop price	quote price < limit price	min(quote price, limit price)
Tick	Quote	Sell	quote price < stop price	quote price > limit price	max(quote price, limit price)
Tick	Trade	Buy	trade price > stop price	trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	trade price < stop price	trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	ask high price > stop price	ask close price < limit price	min(ask high price, limit price)
QuoteBar		Sell	bid low price < stop price	bid close price > limit price	max(bid low price, limit price)
TradeBar		Buy	high price > stop price	close price < limit price	min(high price, limit price)
TradeBar		Sell	low price < stop price	close price > limit price	max(low price, limit price)

The model won't fill stop limit orders with stale data or data with the order timestamp to avoid look-ahead bias.

Trailing Stop Orders

The following table describes the fill logic of trailing stop orders for each data format and order direction. Once the stop condition is met, the model fills the orders and sets the fill price.

Data Format	TickType	Order Direction	Stop Condition	Fill Price
Tick	Quote	Buy	quote price > stop price	max(stop price, quote price + slippage)
Tick	Quote	Sell	quote price < stop price	min(stop price, quote price - slippage)
Tick	Trade	Buy	trade price > stop price	max(stop price, last trade price + slippage)
Tick	Trade	Sell	trade price < stop price	min(stop price, last trade price - slippage)
QuoteBar		Buy	ask high price > stop price	max(stop price, ask close price + slippage)
QuoteBar		Sell	bid low price < stop price	min(stop price, bid close price - slippage)
TradeBar		Buy	high price > stop price	max(stop price, close price + slippage)
TradeBar		Sell	low price < stop price	min(stop price, close price - slippage)

While the stop condition is not met, the model updates the stop price under certain conditions. The following table shows the update condition and stop price value for currency-based trailing amounts:

Data Format	TickType	Order Direction	Update Condition	Stop Price
Tick	Quote	Buy	stop price - quote price <= trailing amount	quote price + trailing amount
Tick	Quote	Sell	quote price - stop price <= trailing amount	quote price - trailing amount
Tick	Trade	Buy	stop price - trade price <= trailing amount	trade price + trailing amount
Tick	Trade	Sell	trade price - stop price <= trailing amount	trade price - trailing amount
QuoteBar		Buy	stop price - ask low price <= trailing amount	ask low price + trailing amount
QuoteBar		Sell	bid high price - stop price <= trailing amount	bid high price - trailing amount
TradeBar		Buy	stop price - low price <= trailing amount	low price + trailing amount
TradeBar		Sell	high price - stop price <= trailing amount	high price - trailing amount

The following table shows the update condition and stop price value for percentage-based trailing amounts.

Data Format	TickType	Order Direction	Update Condition	Stop Price
Tick	Quote	Buy	stop price - quote price <= quote price * trailing amount	quote price * (1 + trailing amount)
Tick	Quote	Sell	quote price - stop price <= quote price * trailing amount	quote price * (1 - trailing amount)
Tick	Trade	Buy	stop price - trade price <= trade price * trailing amount	trade price * (1 + trailing amount)
Tick	Trade	Sell	trade price - stop price <= trade price * trailing amount	trade price * (1 - trailing amount)
QuoteBar		Buy	stop price - ask low price <= ask low price * trailing amount	ask low price * (1 + trailing amount)
QuoteBar		Sell	bid high price - stop price <= bid high price * trailing amount	bid high price * (1 - trailing amount)
TradeBar		Buy	stop price - low price <= stop price * trailing amount	low price * (1 + trailing amount)
TradeBar		Sell	high price - stop price <= high price * trailing amount	high price * (1 - trailing amount)

The model only fills trailing stop orders when the exchange is open.

The model won't fill trailing stop orders with stale data or data with the order timestamp to avoid look-ahead bias.

Market on Open Orders

The following table describes the fill price of market on open orders for each data format and order side:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	quote price + slippage
Tick	Quote	Sell	quote price - slippage
Tick	Trade	Buy	trade price + slippage
Tick	Trade	Sell	trade price - slippage
QuoteBar		Buy	ask open price + slippage
QuoteBar		Sell	bid open price - slippage
TradeBar		Buy	open price + slippage
TradeBar		Sell	open price - slippage

The model won't fill market on open orders during pre-market hours.

The model won't fill market on open orders with stale data or data with the order timestamp to avoid look-ahead bias.

Market on Close Orders

The following table describes the fill price of market on close orders for each data format and order side:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	quote price + slippage
Tick	Quote	Sell	quote price - slippage
Tick	Trade	Buy	trade price + slippage
Tick	Trade	Sell	trade price - slippage
QuoteBar		Buy	ask close price + slippage
QuoteBar		Sell	bid close price - slippage
TradeBar		Buy	close price + slippage
TradeBar		Sell	close price - slippage

The model won't fill market on close orders with stale data or data with the order timestamp to avoid look-ahead bias.

Combo Market Orders

The following table describes the fill price of combo market orders for each data format and order direction:

Data Format	TickType	Order Direction	Fill Price
Tick	Quote	Buy	Ask quote price + slippage
Tick	Quote	Sell	Bid quote price - slippage
Tick	Trade	Buy	Trade price + slippage
Tick	Trade	Sell	Trade price - slippage
QuoteBar		Buy	Ask close price + slippage
QuoteBar		Sell	Bid close price - slippage
TradeBar		Buy	Close price + slippage
TradeBar		Sell	Close price - slippage

The model only fills combo market orders if all the following conditions are met:

- The exchange is open
- The data isn't stale
- All the legs can fill in the same time step after the order time step

The fill quantity of each leg is the product of the leg order quantity and the combo market order quantity.

Combo Limit Orders

To fill combo limit orders, the fill model calculates the aggregate price of the combo order, which is the sum of prices for each security in the order legs. The price of each security is a function of the data format and order direction. Legs with a positive order quantity increase the aggregate price and legs with a negative quantity decrease the aggregate price. The following table shows how the fill model calculates the security prices.

Data Format	TickType	Combo Order Direction	Leg Order Direction	Price
Tick	Quote	Buy or sell	Buy	Ask price
Tick	Quote	Buy or sell	Sell	Bid price
Tick	Trade	Buy or sell	Buy or sell	Trade price
QuoteBar		Buy	Buy	Ask low price
QuoteBar		Buy	Sell	Bid low price
QuoteBar		Sell	Buy	Ask high price
QuoteBar		Sell	Sell	Bid high price
TradeBar		Buy	Buy or sell	Low price
TradeBar		Sell	Buy or sell	High price

After the fill model calculates the aggregate price of the combo order, it checks if it should fill the order. The following table describes the fill condition of the combo order and the fill price price of each leg:

Data Format	TickType	Combo Order Direction	Fill Condition	Leg Order Direction	Fill Price
Tick	Quote	Buy	Aggregate price < combo limit price	Buy or sell	Quote price
Tick	Quote	Sell	Aggregate price > combo limit price	Buy or sell	Quote price
Tick	Trade	Buy	Aggregate price < combo limit price	Buy or sell	Trade price
Tick	Trade	Sell	Aggregate price > combo limit price	Buy or sell	Trade price
QuoteBar		Buy	Aggregate price < combo limit price	Buy	Ask low price
QuoteBar		Buy	Aggregate price < combo limit price	Sell	Bid low price
QuoteBar		Sell	Aggregate price > combo limit price	Buy	Ask high price
QuoteBar		Sell	Aggregate price > combo limit price	Sell	Bid high price
TradeBar		Buy	Aggregate price < combo limit price	Buy or sell	Low price
TradeBar		Sell	Aggregate price > combo limit price	Buy or sell	High price

The model only fills combo limit orders if the data isn't stale and all the legs can fill in the same time step after the order time step. The fill quantity of each leg is the product of the leg order quantity and the combo order quantity.

Combo Leg Limit Orders

The following table describes the fill logic of combo leg limit orders for each data format and order direction. The order direction in the table represents the order direction of the order leg, not the order direction of the combo order.

Data Format	TickType	Order Direction	Fill Condition	Fill Price
Tick	Quote	Buy	Ask price < limit price	min(ask price, limit price)
Tick	Quote	Sell	Bid price > limit price	max(bid price, limit price)
Tick	Trade	Buy	Trade price < limit price	min(trade price, limit price)
Tick	Trade	Sell	Trade price > limit price	max(trade price, limit price)
QuoteBar		Buy	Ask low price < limit price	min(ask high price, limit price)
QuoteBar		Sell	Bid high price > limit price	max(bid low price, limit price)
TradeBar		Buy	Low price < limit price	min(high price, limit price)
TradeBar		Sell	High price > limit price	max(low price, limit price)

The model only fills combo leg limit orders if all the following conditions are met:

- The exchange is open
- The data isn't stale
- All the legs can fill in the same time step after the order time step

The fill quantity is the product of the leg order quantity and the combo order quantity.

Reality Modeling

Slippage

Slippage

Key Concepts

Introduction

Slippage is the difference between the fill price you expect to get for an order and the actual fill price. Since the price can move in the direction of your trade or against the direction of your trade while you wait for the order to fill, slippage can be positive or negative. Slippage models model slippage to make backtest results more realistic.

Factors Impacting Slippage

There are many factors that can impact slippage, including the trading engine, brokerage connection, and market dynamics.

Trading Engine

How long does it take from when you place an order to when it's sent to the brokerage? Longer delays lead to more slippage.

Brokerage Connection

How long does it take for your brokerage to receive an order that you send? Slow internet connections, long travel distances, and poor infrastructure lead to more slippage

Market Dynamics

How volatile is the current market environment? More volatility leads to more slippage.

Does the market consist of sophisticated microsecond arbitrageurs? If your order creates an arbitrage opportunity, it can cause more slippage.

Set Models

The brokerage model of your algorithm automatically sets the slippage model for each security, but you can override it. To manually set the slippage model of a security, call the `SetSlippageModel` method on the `Security` object.

C#

```
public override void Initialize()
{
    var security = AddEquity("SPY");
    // Set the slippage model for the requested security to backtest with the most realistic scenario
    // VolumeShareSlippageModel has slippage size affected by the volume of the order compared to the actual filled
volume of the bar
    // It is only valid for the security with a volume property, while being more accurate with denser resolution
    security.SetSlippageModel(new VolumeShareSlippageModel());
}
```

You can also set the slippage model in a security initializer . If your algorithm has a dynamic universe, use the security initializer

technique. In order to initialize single security subscriptions with the security initializer, call `AddSecurityInitializer` before you create the subscriptions.

```C#
public class AddSecurityInitializerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the security initializer to set models of assets.
        AddSecurityInitializer(CustomSecurityInitializer);
    }

    private void CustomSecurityInitializer(Security security)
    {
        // Overwrite some of the reality models
        security.SetSlippageModel(new VolumeShareSlippageModel());
    }
}
```

To view all the pre-built slippage models, see Supported Models .

Default Behavior

The brokerage model of your algorithm automatically sets the slippage model of each security. The default brokerage model is the `DefaultBrokerageModel` , which uses the NullSlippageModel to model zero slippage for all securities.

Model Structure

Slippage models should implement the `ISlippageModel` interface. Extensions of the `ISlippageModel` interface must implement the `GetSlippageApproximation` method, which calculates the slippage quantity.

```C#
public class CustomSlippageModelExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the custom slippage model for an added security to use the custom model
        var security = AddEquity("SPY");
        security.SetSlippageModel(new MySlippageModel());
    }
}

// Define the custom slippage model
public class MySlippageModel : ISlippageModel
{
    public decimal GetSlippageApproximation(Security asset, Order order)
    {
        return asset.Price*0.0001m*(decimal) Math.Log10(2*(double) order.AbsoluteQuantity);
    }
}
```

Examples

The following examples demonstrate some common practices for implementing a custom slippage model.

Example 1: Logarithmic Scaling

The following algorithm defines a custom slippage model where the slippage increases logarithmically with order size, scaled by 0.01% of the asset price, modeling realistic market impact where larger orders have diminishing marginal price impact.

```C#
public class CustomSlippageModelAlgorithm : QCAlgorithm
{

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var security = AddEquity("SPY", Resolution.Daily);
        security.SetSlippageModel(new MySlippageModel());
    }

    public override void OnData(Slice data)
    {
        if (!Portfolio.Invested)
        {
            SetHoldings("SPY", 1);
        }
    }
}

public class MySlippageModel : ISlippageModel
{
    public decimal GetSlippageApproximation(Security asset, Order order)
    {
        return asset.Price*0.0001m*(decimal) Math.Log10(2*(double) order.AbsoluteQuantity);
    }
}
```

Example 2: Volume Ratio Capped Slippage Model

The following algorithm trades the microeconomic demand popularity gap between the top 10 and bottom 10 liquid constituents of QQQ. A custom slippage model simulates the slippage on less liquid stock fills. The slippage percent is set to be 2% multiplied by the ratio of the order quantity to the previous minute bar's volume, maximized at 2%.

C#

```
public class RealityModelingSlippageAlgorithm : QCAlgorithm
{
    private List<Symbol> _longs = new();
    private List<Symbol> _shorts = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        Settings.SeedInitialPrices = true;
        AddSecurityInitializer(security =>
        {
            // To set the slippage model to the custom one for all stocks entering the universe.
            security.SetSlippageModel(new CustomSlippageModel());
        });

        // Request extended market hour QQQ data for trading.
        var qqq = AddEquity("QQQ").Symbol;

        // Weekly portfolio updating to allow time to capitalize on the popularity gap.
        UniverseSettings.Schedule.On(DateRules.WeekStart());
        // Add universe to trade on the most and least liquid stocks among QQQ constituents.
        AddUniverse(
            // First, we select all QQQ constituents for the next filter on liquidity.
            Universe.ETF(qqq, Market.USA, UniverseSettings, (constituents) => constituents.Select(c => c.Symbol)),
            FundamentalSelection
        );

        // Set a scheduled event to rebalance the portfolio at the start of every week.
        Schedule.On(
            DateRules.WeekStart(qqq),
            TimeRules.AfterMarketOpen(qqq),
            Rebalance
        );
    }

    private IEnumerable<Symbol> FundamentalSelection(IEnumerable<Fundamental> fundamentals)
    {
        var sortedByDollarVolume = fundamentals.OrderBy(x => x.DollarVolume).ToList();
        // Add the 10 most liquid stocks to the universe to long later.
        _longs = sortedByDollarVolume.TakeLast(10)
            .Select(x => x.Symbol)
            .ToList();
        // Add the 10 least liquid stocks to the universe to short later.
        _shorts = sortedByDollarVolume.Take(10)
            .Select(x => x.Symbol)
            .ToList();

        return _longs.Union(_shorts);
    }

    private void Rebalance()
    {
        // Equally invest in the selected stocks to dissipate capital risk evenly.
        // Dollar neutral of long and short stocks to eliminate systematic risk, only capitalize the popularity gap.
        var targets = _longs.Select(symbol => new PortfolioTarget(symbol, 0.05m)).ToList();
        targets.AddRange(_shorts.Select(symbol => new PortfolioTarget(symbol, -0.05m)).ToList());

        // Liquidate the stocks that are not the most and least popular to release funds for trades with higher
expected returns.
        SetHoldings(targets, liquidateExistingHoldings: true);
    }

    internal class CustomSlippageModel : ISlippageModel
    {
        public decimal GetSlippageApproximation(Security asset, Order order)
        {
            // We set a maximum 2% slippage that is linearly linked to the ratio of the order size versus the
previous bar's volume.
            return asset.Price * 0.02m * Math.Min(1m, order.AbsoluteQuantity / asset.Volume);
        }
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
CustomModelsAlgorithm.cs C#

Slippage

Supported Models

Introduction

This page describes the pre-built slippage models in LEAN. If none of these models perform exactly how you want, create a custom slippage model .

Null Model

The `NullSlippageModel` sets the slippage of each order to zero. It's the default slippage model of the DefaultBrokerageModel .

```
C#
security.SetSlippageModel(NullSlippageModel.Instance);
```

For more information about this model, see the class reference and implementation .

Constant Model

The `ConstantSlippageModel` applies a constant percentage of slippage to each order.

```
C#
security.SetSlippageModel(new ConstantSlippageModel(0.01m));
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`slippagePercent`	`decimal`	The slippage percent for each order. The value must be in the interval (0, 1).	

For more information about this model, see the class reference and implementation .

Volume Share Model

The `VolumeShareSlippageModel` calculates the slippage of each order by multiplying the price impact constant by the square of the ratio of the order to the total volume. If the volume of the current bar is zero, the slippage percent is

$$volumeLimit^2 * priceImpact$$

where volumeLimit and priceImpact are custom input variables. If the volume of the current bar is positive, the slippage percent is

$$\min(\frac{|orderQuantity|}{barVolume}, volumeLimit)^2 * priceImpact$$

where orderQuantity is the quantity of the order and barVolume is the volume of the current bar. If the security subscription provides `TradeBar` data, the barVolume is the volume of the current bar. If the security subscription provides `QuoteBar` data, the barVolume is the bid or ask size of the current bar. CFD, Forex, and Crypto data often doesn't include volume. If there is no

volume reported for these asset classes, the model returns zero slippage.

```
C#
security.SetSlippageModel(new VolumeShareSlippageModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`volumeLimit`	`decimal`	Maximum percent of historical volume that can fill in each bar. 0.5 means 50% of historical volume. 1.0 means 100%.	0.025
`priceImpact`	`decimal`	Scaling coefficient for price impact. Larger values will result in more simulated price impact. Smaller values will result in less simulated price impact.	0.1

For more information about this model, see the class reference and implementation .

Market Impact Model

The `MarketImpactSlippageModel` calculates the slippage with perspective to the market impact created by an order, which mimics the consumption of the order book and disruption of the supply-demand relationship at the microeconomic scale. The model is divided into distinct parts, the market impact estimation and the slippage that results from it.

Factors to Consider

The model includes factors that create market impact. Almgren, Thum, Hauptmann & Li (2005) suggested the following factors:

1. Execution Time

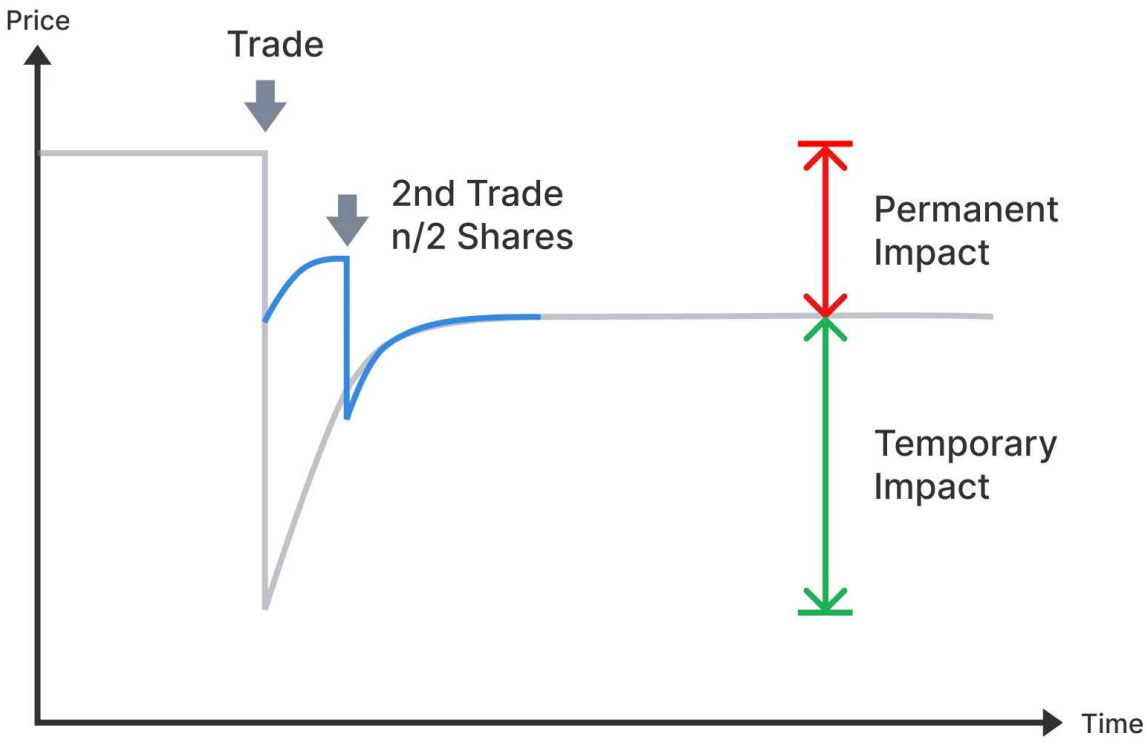
   Asset price time series is usually considered as Random walk (with/without drift) $S_t = S_{t-1} + f(S, t) + \epsilon$ or Brownian Motion $S_t = S_0 + exp((\mu-\frac{1}{2}\sigma^2)t + \sigma W_t)$ In either model, the variance of predictions increases over time ($Var(S)=\sigma\sqrt{t}$). In fact, this happens to most stock price models. Thus, time uncertainty should also apply to the concept of slippage. The longer it takes to go from order submission to execution, the further the fill price will be from the price at order submission.

2. Volatility of the Underlying Securities

   Like (1), the volatility also contributes to the uncertainty of the next price ($Var(S)=\sigma\sqrt{t}$). In most cases, volatility has a one-sided impact since the market's volatility is often introduced by overbuying or overselling from positive or negative sentiments.

3. Liquidity

   Liquidity plays a crucial role in the supply-demand resilience. When a stock is liquid, an order usually only takes a small fraction of the market depth. Hence, the gap between the bid and ask price can be refilled quickly due to high market participation and a small gap. However, when a stock is illiquid, a single order might already take the top few layers of the order book, creating a wide gap between the bid and ask price and volume, which makes a bigger shift in the equilibrium price. This will create an impact which is permanent in nature.

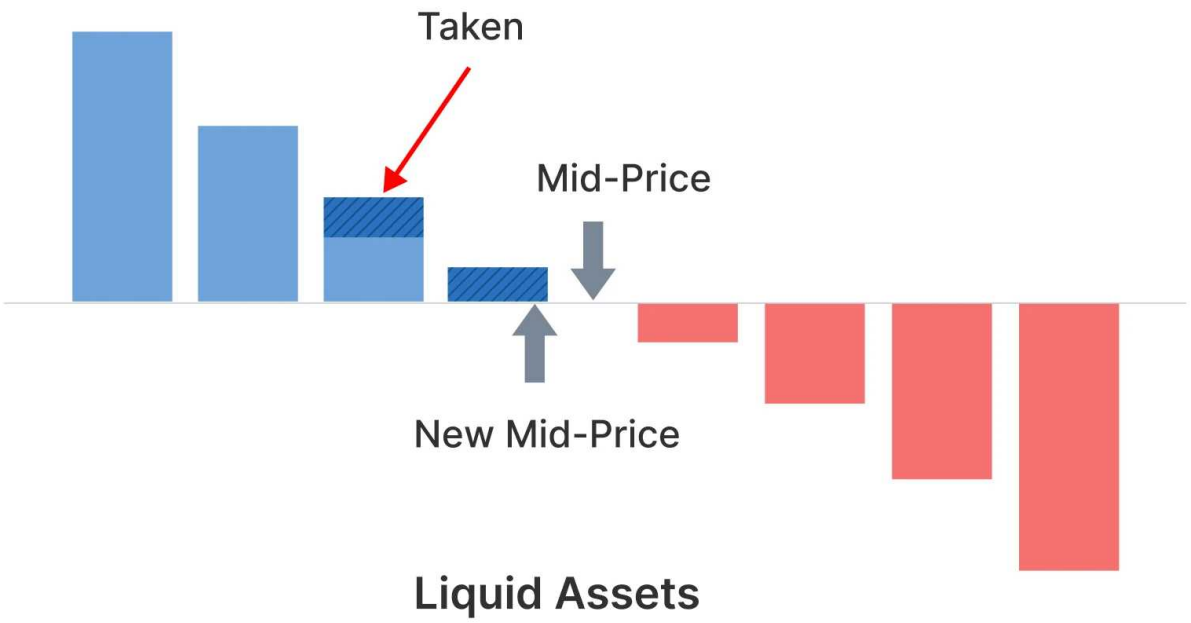


4. Order Size

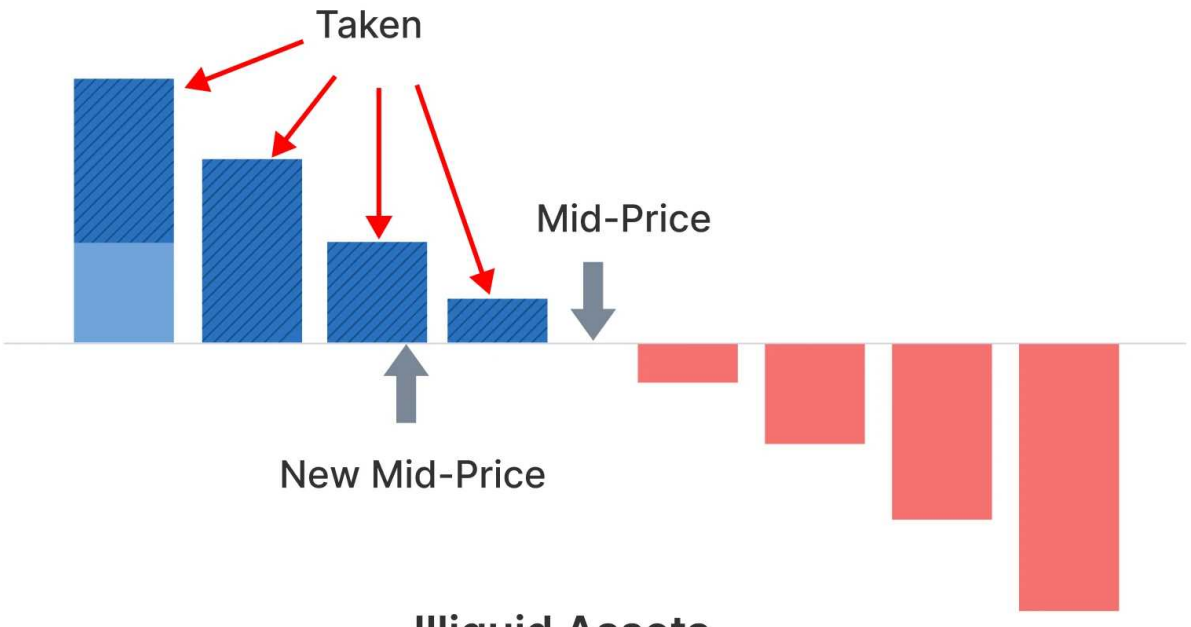
   Last but not least, the order size directly affects the proportion of the order book/liquidity that the order consumes. Moreover, the larger the order is, the longer it takes to fill since the market might not have sufficient depth. This has interaction effects with the preceding factors.

Market Impact Decomposition

In the book "A Signal Processing Perspective on Financial Engineering", Feng & Palomer (2015) generalized scholars' opinions that market impact can be broken down into "permanent" and "temporary" components, with noise. The permanent impact of a trade refers to the price differences between equilibrium prices before and after the trade and the resilience of the order book. In contrast, temporary impact refers to the price difference between the instant price drop after a trade and the post-trade equilibrium.



Liquid Assets



Illiquid Assets

The model incorporates the preceding factors and estimates both the permanent and temporary impact components with two diffusion models as a function of $\nu=\frac{X}{VT}$, which is the normalized order size:

$G\left(\nu\right)=\pm\gamma\left|\nu\right|^\alpha+\text{noise}$ $H\left(\nu\right)=\pm\eta\left|\nu\right|^\beta+\text{noise}$

where X is the order quantity, V is the average daily volume, and T is the time difference between order submission and filling (execution time). The other symbols are parameters that are calibrated from historical trade data. Together with the execution time, daily volatility σ, and liquidity factor δ, the complete models are as follows: $I=\sigma T \gamma\left|\frac{X}$

{VT}\right|^\alpha\left(\frac{\Theta}{V}\right)^\delta+\sigma\sqrt{T_{\text{post}}}\epsilon J=\sigma T \eta\left|\frac{X}{VT}\right|^\beta+\sigma\sqrt{T_{\text{post}}}\epsilon where \Theta is the share outstanding, T_{post} is the time difference between order submission and the new equilibrium established, I is the permanent market impact, and J is the temporary market impact. The inverse of the turnover rate \frac{\Theta}{V} contributes to the permanent impact. This relationship is reasonable since higher turnover means more market participants are in the market, so the order book depth is deeper.

Implementation

```C#
security.SetSlippageModel(new MarketImpactSlippageModel(self));
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`algorithm`	`IAlgorithm`	The algorithm instance	
`nonNegative`	`bool`	Indicator whether only non-negative slippage allowed.	`true`
`latency`	`double`	Seconds between order submission and order fill	0.075
`impactTime`	`double`	Seconds between order fill and the establishment of the new equilibrium	1800
`alpha`	`double`	Exponent of the permanent impact function	0.891
`beta`	`double`	Exponent of the temporary impact function	0.600
`gamma`	`double`	Coefficient of the permanent impact function	0.314
`eta`	`double`	Coefficient of the temporary impact function	0.142
`delta`	`double`	Liquidity scaling factor for permanent impact	0.267
`randomSeed`	`int`	Random seed for generating gaussian noise	50

The default parameters of the `MarketImpactSlippageModel` are the parameters that the authors calibrated and shared. However, you may want to reference the original paper and recalibrate the parameters because of the following reasons:

- The default parameters are calibrated around 2 decades ago
- The trading time effect is not accounted (volume near market open/close is larger)
- The market environment does not have many market makers at that time

For more information about this model, see the class reference and implementation .

Reality Modeling

Transaction Fees

Transaction Fees

Key Concepts

Introduction

Your orders incur a transaction fee when a brokerage fills them in the market. LEAN uses transaction fee models in backtesting to model the live trading fees you would incur with the strategy. Transaction fee models make backtest results more realistic. To give your backtests the most accurate fees, LEAN contains transaction fee models that model the fee structure of many popular brokerages.

Set Models

The brokerage model of your algorithm automatically sets the fee model for each security, but you can override it. To manually set the fee model of a security, call the `SetFeeModel` method on the `Security` object.

C#

```
public override Initialize()
{
    var security = AddEquity("SPY");
    // Set the fee model for the requested security to backtest with the most realistic scenario
    // Constant fee model of 0 dollar is accurate for the commission-free brokerage
    security.SetFeeModel(new ConstantFeeModel(0));
}
```

You can also set the fee model in a security initializer . If your algorithm has a dynamic universe, use the security initializer technique. In order to initialize single security subscriptions with the security initializer, call `AddSecurityInitializer` before you create the subscriptions.

C#

```
public class AddSecurityInitializerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the security initializer to set models of assets.
        AddSecurityInitializer(CustomSecurityInitializer);
    }

    private void CustomSecurityInitializer(Security security)
    {
        // Overwrite some of the reality models
        security.SetFeeModel(new ConstantFeeModel(0));
    }
}
```

In live trading, the `SetFeeModel` method isn't ignored. If we use order helper methods like `SetHoldings` , the fee model helps to calculate the order quantity. However, the algorithm doesn't update the cash book with the fee from the fee model. The algorithm uses the actual fee from the brokerage to update the cash book.

To view all the pre-built fee models, see Supported Models .

Default Behavior

The brokerage model of your algorithm automatically sets the fill model for each security. The default brokerage model is the `DefaultBrokerageModel` , which set the ConstantFeeModel with no fees for Forex, CFD, and Crypto assets and sets the InteractiveBrokersFeeModel for the remaining asset classes.

Model Structure

Fee models should extend the `FeeModel` class. Extensions of the `FeeModel` class must implement the `GetOrderFee` method, which receives `OrderFeeParameters` and returns an `OrderFee` that represents a cash amount in a currency.

```
C#

public class CustomFeeModelExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the custom fee model for an added security to use the custom model
        var security = AddEquity("SPY");
        security.SetFeeModel(new MyFeeModel());
    }
}

// Define the custom fee model
public class MyFeeModel : FeeModel
{
    public override OrderFee GetOrderFee(OrderFeeParameters parameters)
    {
        return new OrderFee(new CashAmount(0.5m, "USD"));
    }
}
```

The `OrderFeeParameters` object has the following members:

Zero Transaction Fees

To model zero transaction fees, use the ConstantFeeModel with zero as `fee` and specify the `currency` .

```
C#

public override Initialize()
{
    var security = AddEquity("SPY");
    security.SetFeeModel(new ConstantFeeModel(fee: 0m, currency: "USD"));
}
```

Negative Transaction Fees

If you short a security and receive interest payments, they are negative transaction fees.

Examples

The following examples demonstrate common practices for implementing a custom fee model.

Example 1: Fixed Fees

The following algorithm incurs a $0.50 fee for each order.

```
C#
public class CustomFeeModelAlgorithm : QCAlgorithm
{

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        var security = AddEquity("SPY", Resolution.Daily);
        security.SetFeeModel(new MyFeeModel());
    }

    public override void OnData(Slice data)
    {
        if (!Portfolio.Invested)
        {
            SetHoldings("SPY", 1);
        }
    }
}

public class MyFeeModel : FeeModel
{
    public override OrderFee GetOrderFee(OrderFeeParameters parameters)
    {
        return new OrderFee(new CashAmount(0.5m, "USD"));
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
CustomModelsAlgorithm.cs C# RawPricesCoarseUniverseAlgorithm.cs C#

Transaction Fees

Supported Models

Introduction

This page describes some of the pre-built fee models in LEAN. For more brokerage-specific fee models, see the brokerage model documentation . If none of these models perform exactly how you want, create a custom fee model .

Constant Model

The `ConstantFeeModel` applies the absolute value of a constant fee to each order. It's the default fee model of the DefaultBrokerageModel if you trade Forex, CFD, or Crypto assets. Use this model to set zero transaction fees .

```
C#
security.SetFeeModel(new ConstantFeeModel(0.05m));
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`fee`	`decimal`	The order fee quantity	
`currency`	`string`	The order fee currency	"USD"

For more information about this model, see the class reference and implementation .

Alpaca Model

The `AlpacaFeeModel` models the fees of Alpaca .

```
C#
security.SetFeeModel(new AlpacaFeeModel());
```

The `AlpacaFeeModel` charges zero fees for Equity and Equity Options trading. For Crypto trades, it charges a 0.15% maker and 0.25% taker fee. If you add liquidity to the order book by placing a limit order that doesn't cross the spread, you pay maker fees. If you remove liquidity from the order book by placing an order that crosses the spread, you pay taker fees. Alpaca adjusts your fees based on your 30-day trading volume, but we don't currently model these metrics to adjust fees.

The `AlpacaFeeModel` charges fees in the currency you receive from a trade. For example, if you buy ETHBTC, you pay fees in ETH. If you sell ETHBTC, you pay fees in BTC.

For more information about this model, see the class reference and implementation .

Binance Model

The `BinanceFeeModel` models the fees of spot trades on Binance and Binance US .

```
C#
security.SetFeeModel(new BinanceFeeModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`mFee`	`decimal`	Maker fee value	0.001
`tFee`	`decimal`	Taker fee value	0.001

The `BinanceFeeModel` charges the order fees of Binance and Binance US at the VIP 0 level, which is a 0.1% maker and taker fee. If you add liquidity to the order book by placing a limit order that doesn't cross the spread, you pay maker fees. If you remove liquidity from the order book by placing an order that crosses the spread, you pay taker fees. Binance adjusts your fees based on your 30-day trading volume and BNB balance, but we don't currently model these metrics to adjust fees.

The `BinanceFeeModel` charges fees in the currency you receive from a trade. For example, if you buy ETHBTC, you pay fees in ETH. If you sell ETHBTC, you pay fees in BTC.

For more information about this model, see the class reference and implementation .

Binance Futures Model

The `BinanceFuturesFeeModel` models the Binance Futures live trading fees .

```
C#
security.SetFeeModel(new BinanceFuturesFeeModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`mUsdtFee`	`decimal`	Maker fee value for USDT pair contracts	0.0002
`tUsdtFee`	`decimal`	Taker fee value for USDT pair contracts	0.0004
`mBusdFee`	`decimal`	Maker fee value for BUSD pair contracts	0.00012
`tBusdFee`	`decimal`	Taker fee value for BUSD pair contracts	0.00036

By default, the `BinanceFuturesFeeModel` charges the order fees of Binance Futures at the Regular User level. If you add liquidity to the order book by placing a limit order that doesn't cross the spread, you pay maker fees. If you remove liquidity from the order book by placing an order that crosses the spread, you pay taker fees. Binance adjusts your fees based on your 30-day trading volume and BNB balance, but we don't currently model these metrics to adjust fees.

The `BinanceFuturesFeeModel` charges fees in the currency you receive from a trade. For example, if you buy BTCUSD, you pay fees in BTC. If you sell BTCUSD, you pay fees in USD.

For more information about this model, see the class reference and implementation .

Binance Coin Futures Model

The `BinanceCoinFuturesFeeModel` models the Binance Coin Futures live trading fees .

```
security.SetFeeModel(new BinanceCoinFuturesFeeModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`mFee`	`decimal`	Maker fee value	0.0001
`tFee`	`decimal`	Taker fee value	0.0005

By default, the `BinanceCoinFuturesFeeModel` charges the order fees of Binance Coin Futures at the Regular User level. If you add liquidity to the order book by placing a limit order that doesn't cross the spread, you pay maker fees. If you remove liquidity from the order book by placing an order that crosses the spread, you pay taker fees. Binance adjusts your fees based on your 30-day trading volume and BNB balance, but we don't currently model these metrics to adjust fees.

The `BinanceCoinFuturesFeeModel` charges fees in the currency you receive from a trade. For example, if you buy BTCUSDT, you pay fees in BTC. If you sell BTCUSDT, you pay fees in USDT.

For more information about this model, see the class reference and implementation .

Bitfinex Model

The `BitfinexFeeModel` models the Bitfinex live trading fees .

```
security.SetFeeModel(new BitfinexFeeModel());
```

The `BitfinexFeeModel` charges a 0.1% maker fee and a 0.2% taker fee. If you place a limit order that hits a hidden order or you add liquidity to the order book by placing a limit order that doesn't cross the spread, you pay maker fees. If you place a hidden order or you remove liquidity from the order book by placing an order that crosses the spread, you pay taker fees. Bitfinex adjusts your fees based on your 30-day trading volume and LEO balance, but we don't currently model these metrics to adjust fees.

The `BitfinexFeeModel` charges fees in the currency you receive from a trade. For example, if you buy ETHBTC, you pay fees in ETH. If you sell ETHBTC, you pay fees in BTC.

For more information about this model, see the class reference and implementation .

Bybit Model

The `BybitFeeModel` models the Bybit live trading fees .

```
security.SetFeeModel(new BybitFeeModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
mFee	decimal	Maker fee value	0.001
tFee	decimal	Taker fee value	0.001

The BybitFeeModel charges the order fees of Bybit at the VIP 0 level, which is a 0.1% maker and taker fee for spot trades. If you add liquidity to the order book by placing a limit order that doesn't cross the spread, you pay maker fees. If you remove liquidity from the order book by placing an order that crosses the spread, you pay taker fees. Bybit adjusts your fees based on your 30-day trading volume, net borrowings, or asset balance, but we don't currently model these metrics to adjust fees. Bybit also charges different fees for fiat-Crypto pairs, (for example, USDT/EUR), but the BybitFeeModel charges the same fee rate for all pairs.

The BybitFeeModel charges fees in the currency you receive from a trade. For example, if you buy ETHBTC, you pay fees in ETH. If you sell ETHBTC, you pay fees in BTC.

For more information about this model, see the class reference and implementation .

Bybit Futures Model

The BybitFuturesFeeModel models the Bybit live trading fees .

```
C#
security.SetFeeModel(new BybitFuturesFeeModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
makerFee	decimal	Maker fee value	0.0002
takerFee	decimal	Taker fee value	0.00055

The BybitFuturesFeeModel charges the order fees of Bybit at the VIP 0 level, which is a 0.02% maker and a 0.055% taker fee for Crypto Futures trades. If you add liquidity to the order book by placing a limit order that doesn't cross the spread, you pay maker fees. If you remove liquidity from the order book by placing an order that crosses the spread, you pay taker fees. Bybit adjusts your fees based on your 30-day trading volume, net borrowings, or asset balance, but we don't currently model these metrics to adjust fees.

The BybitFuturesFeeModel charges fees in the currency you receive from a trade. For example, if you buy BTCUSDT, you pay fees in BTC. If you sell BTCUSDT, you pay fees in USDT.

For more information about this model, see the class reference and implementation .

Charles Schwab Model

The CharlesSchwabFeeModel models the fees of Charles Schwab .

```
C#
security.SetFeeModel(new CharlesSchwabFeeModel());
```

The CharlesSchwabFeeModel charges $0 USD for Equity trades.

For more information about this model, see the class reference and implementation .

Coinbase Model

The `CoinbaseFeeModel` models the Coinbase live trading fees .

```
C#
security.SetFeeModel(new CoinbaseFeeModel());
```

The `CoinbaseFeeModel` models the order fees of Coinbase at the $50K-100K pricing tier for all Crypto pairs, so it charges a 0.6% maker and a 0.8% taker fee for most pairs. The following table shows the Coinbase Stable Pairs, which charge a 0% maker fee and a 0.001% taker fee:

Stable Pairs			
DAIUSDC	DAIUSD	GYENUSD	PAXUSD
PAXUSDT	MUSDUSD	USDCEUR	USDCGBP
USDTEUR	USDTGBP	USDTUSD	USDTUSDC
USTEUR	USTUSD	USTUSDT	WBTCBTC

If you add liquidity to the order book by placing a limit order that doesn't cross the spread, you pay maker fees. If you remove liquidity from the order book by placing an order that crosses the spread, you pay taker fees. Coinbase adjusts your fees based on your 30-day trading volume, but we don't currently model trading volume to adjust fees.

The `CoinbaseFeeModel` models the adjustments Coinbase has made to their fees over time. The following table shows the fees for each time period:

Time Period (UTC)	**Maker Fee (%)**	**Taker Fee (%)**	**Stable Coin**
Time < 3/23/2019 1:30AM	0	0.3	
3/23/2019 1:30AM <= Time < 10/8/2019 12:30AM	0.15	0.25	
10/8/2019 12:30AM <= Time	0.5	0.5	
Time < 6/1/2022 3/23/2019 12:00AM	0	0.1%	✓

The `CoinbaseFeeModel` charges fees in the quote currency .

For more information about this model, see the class reference and implementation .

dYdX Model

The `dYdXFeeModel` models the dYdX live trading fees .

```
C#
security.SetFeeModel(new dYdXFeeModel());
```

The `dYdXFeeModel` models the order fees of dYdX at the lowest pricing tier for all Cryptofuture pairs, so it charges a 0.01% maker and a 0.05% taker fee.

If you add liquidity to the order book by placing a limit order that doesn't cross the spread, you pay maker fees. If you remove liquidity from the order book by placing an order that crosses the spread, you pay taker fees. dYdX adjusts your fees based on your 30-day trading volume, but we don't currently model trading volume to adjust fees.

The `dYdXFeeModel` charges fees in the quote currency .

For more information about this model, see the class reference and implementation .

Interactive Brokers Model

The `InteractiveBrokersFeeModel` models the fees of Interactive Brokers .

```
C#
security.SetFeeModel(new InteractiveBrokersFeeModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`monthlyForexTradeAmountInUSDollars`	`decimal`	Monthly Forex dollar volume traded	0
`monthlyOptionsTradeAmountInContracts`	`decimal`	Monthly Option contracts traded	0

The following table describes which currency the `InteractiveBrokersFeeModel` charges fees in for each asset class:

Asset Class	Fee Currency
US Equity	USD
India Equity	INR
Equity Options	USD
Forex	USD
US Futures	USD
Hong Kong Futures	The contract quote currency (CNH, HKD, or USD)
US Future Options	USD
Hong Kong Future Options	The contract quote currency (CNH, HKD, or USD)
Index Options	USD
CFD	The contract quote currency

The following sections describe the trading fees of each asset class.

For more information about this model, see the class reference and implementation .

US Equities

US Equity trades cost 0.005/share with a 1 minimum fee and a 0.5% maximum fee.

Equity Options and Index Options

Equity Options and Index Options fees are a function of your monthly volume and the premium of the contract you trade. The following table shows the fees for each volume tier:

Monthly Volume (Contacts)	Premium ($)	Fee per Contract ($)
<= 10,000	< 0.05	0.25
<= 10,000	0.05 <= premium < 0.10	0.50
<= 10,000	>= 0.10	0.70
10,000 < volume <= 50,000	< 0.05	0.25
10,000 < volume <= 50,000	>= 0.05	0.50
50,000 < volume <= 100,000	Any	0.25
> 100,000	Any	0.15

By default, LEAN models your fees at the tier with the lowest monthly volume. To adjust the fee tier, manually set the fee model and provide a `monthlyOptionsTradeAmountInContracts` argument.

The following table shows how the model determines the premium for each order type and direction:

Order Type	Buy Order Premium	Sell Order Premium
Market	Bid price	Ask price
Limit	Limit price	Limit price
Limit if touched	Limit price	Limit price
Stop market	Stop price	Stop price
Stop limit	Limit price	Limit price
Trailing stop	Stop price	Stop price
Market on open	Bid price	Ask price
Market on close	Bid price	Ask price
Combo market	Bid price	Ask price
Combo limit	Limit price	Limit price
Combo leg limit	Limit price	Limit price

There is no fee to exercise Option contracts.

Forex

Forex fees are a function of your monthly Forex trading volume. The following table shows the fee tiers:

Monthly Volume (USD)	Commission Rate (%)	Minimum Fee ($)
<= 1B	0.002	2
1B < volume <= 2B	0.0015	1.5
2B < volume <= 5B	0.001	1.25
> 5B	0.0008	1

By default, LEAN models your fees at the tier with the lowest monthly volume. To adjust the fee tier, manually set the fee model and provide a `monthlyForexTradeAmountInUSDollars` argument.

US Futures

US Futures fees depend on the contracts you trade. The following table shows the base fee per contract for each Future:

Contract Symbol	Market	Name	Base Fee Per Contract ($)	Exchange Fee Per Contract ($)
E-mini Futures				
ES	CME	E-mini S&P 500 Futures	0.85	1.28
NQ	CME	E-mini Nasdaq-100 Futures	0.85	1.28
YM	CBOT	E-mini Dow ($5) Futures	0.85	1.28
RTY	CME	E-mini Russell 2000 Index Futures	0.85	1.28
EMD	CME	E-mini S&P MidCap 400 Futures	0.85	1.28
Micro E-mini Futures				
MYM	CBOT	Micro E-mini Dow Jones Industrial Average Index Futures	0.25	0.3
M2K	CME	Micro E-mini Russell 2000 Index Futures	0.25	0.3
MES	CME	Micro E-mini Standard and Poor's 500 Stock Price Index Futures	0.25	0.3
MNQ	CME	Micro E-mini Nasdaq-100 Index Futures	0.25	0.3

2YY	CBOT	Micro 2-Year Yield Futures	0.25	0.3
5YY	CBOT	Micro 5-Year Yield Futures	0.25	0.3
10Y	CBOT	Micro 10-Year Yield Futures	0.25	0.3
30Y	CBOT	Micro 30-Year Yield Futures	0.25	0.3
MCL	NYMEX	Micro WTI Crude Oil Futures	0.25	0.3
MGC	COMEX	Micro Gold Futures	0.25	0.3
SIL	COMEX	Micro Silver Futures	0.25	0.3
Cryptocurrency Futures				
BTC	CME	Bitcoin Futures	5	6
MIB	CME	BTIC on Micro Bitcoin Futures	2.25	2.5
MBT	CME	Micro Bitcoin Futures	2.25	2.5
MET	CME	Micro Ether Futures	0.2	0.2
MRB	CME	BTIC on Micro Ether Futures	0.2	0.2
E-mini FX Futures				
E7	CME	E-mini Euro FX Futures	0.5	0.85
J7	CME	E-mini Japanese Yen Futures	0.5	0.85
Micro E-mini FX Futures				
M6E	CME	Micro Euro/U.S. Dollar (EUR/USD) Futures	0.15	0.24
M6A	CME	Micro Australian Dollar/U.S. Dollar (AUD/USD) Futures	0.15	0.24
M6B	CME	Micro British Pound Sterling/U.S. Dollar (GBP/USD) Futures	0.15	0.24
MCD	CME	Micro Canadian Dollar/U.S.Dollar(CAD/USD) Futures	0.15	0.24

MJY	CME	Micro Japanese Yen/U.S. Dollar (JPY/USD) Futures	0.15	0.24
MSF	CME	Micro Swiss Franc/U.S. Dollar (CHF/USD) Futures	0.15	0.24
M6J	CME	Micro USD/JPY Futures	0.15	0.24
MIR	CME	Micro INR/USD Futures	0.15	0.24
M6C	CME	Micro USD/CAD Futures	0.15	0.24
M6S	CME	Micro USD/CHF Futures	0.15	0.24
MNH	CME	Micro USD/CNH Futures	0.15	0.24

If you trade a contract that's not in the preceding table, the base fee is 0.85/contract and the exchange fee is 1.60/contract.

In addition to the base fee and exchange fee, there is a $0.02/contract regulatory fee.

Futures Options

Futures Options fees depend on the contracts you trade. The following table shows the base fee per contract for each Future:

Contract Symbol	**Market**	**Underlying Futures Name**	**Base Fee Per Contract ($)**	**Exchange Fee Per Contract ($)**
E-mini Futures Options				
ES	CME	E-mini S&P 500 Futures	0.85	0.55
NQ	CME	E-mini Nasdaq-100 Futures	0.85	0.55
YM	CBOT	E-mini Dow ($5) Futures	0.85	0.55
RTY	CME	E-mini Russell 2000 Index Futures	0.85	0.55
EMD	CME	E-mini S&P MidCap 400 Futures	0.85	0.55
Micro E-mini Futures Options				
MYM	CBOT	Micro E-mini Dow Jones Industrial Average Index Futures	0.25	0.2
M2K	CME	Micro E-mini Russell 2000 Index Futures	0.25	0.2

MES	CME	Micro E-mini Standard and Poor's 500 Stock Price Index Futures	0.25	0.2
MNQ	CME	Micro E-mini Nasdaq-100 Index Futures	0.25	0.2
2YY	CBOT	Micro 2-Year Yield Futures	0.25	0.2
5YY	CBOT	Micro 5-Year Yield Futures	0.25	0.2
10Y	CBOT	Micro 10-Year Yield Futures	0.25	0.2
30Y	CBOT	Micro 30-Year Yield Futures	0.25	0.2
MCL	NYMEX	Micro WTI Crude Oil Futures	0.25	0.2
MGC	COMEX	Micro Gold Futures	0.25	0.2
SIL	COMEX	Micro Silver Futures	0.25	0.2
Cryptocurrency Futures Options				
BTC	CME	Bitcoin Futures	5	5
MIB	CME	BTIC on Micro Bitcoin Futures	1.25	2.5
MBT	CME	Micro Bitcoin Futures	1.25	2.5
MET	CME	Micro Ether Futures	0.1	0.2
MRB	CME	BTIC on Micro Ether Futures	0.1	0.2

If you trade a contract that's not in the preceding table, the base fee is 0.85/contract and the exchange fee is 1.60/contract.

In addition to the base fee and exchange fee, there is a $0.02/contract regulatory fee.

CFD

CFD trades cost 0.002% of the trade value. The following table shows the minimum fee for each quote currency:

Quote Currency	**Fee**
JPY	¥40
MKD	HK$10
Other	1

CFD trading fees are denominated in the quote currency of the CFD product.

Kraken Model

The `KrakenFeeModel` models the fees of Kraken .

```
C#
security.SetFeeModel(new KrakenFeeModel());
```

The `KrakenFeeModel` model the order fees of Kraken. For trading pairs that contain only Crypto assets, this fee model models the lowest tier in Kraken's tiered fee structure, which is a 0.16% maker fee and a 0.26% taker fee. If you add liquidity to the order book by placing a limit order that doesn't cross the spread, you pay maker fees. If you remove liquidity from the order book by placing an order that crosses the spread, you pay taker fees. For trading pairs that have any of the following currencies as the base currency in the pair, the fee is 0.2%:

- CAD
- EUR
- GBP
- JPY
- USD
- USDT
- DAI
- USDC

The `KrakenFeeModel` charges fees in the quote currency for buy orders and in the base currency for sell orders. To override these settings, define the `KrakenOrderProperties` .

```
C#
public override void Initialize()
{
    // Set the default order properties
    DefaultOrderProperties = new KrakenOrderProperties()
    {
        FeeInBase = true,
        FeeInQuote = false
    };
}

public override void OnData(Slice slice)
{
    // Override the default order properties
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new KrakenOrderProperties
               {
                   FeeInQuote = true,
                   FeeInBase = false
               });
}
```

Kraken adjusts your fees based on your 30-day trading volume, but the `KrakenFeeModel` don't currently model trading volume to adjust fees.

For more information about this model, see the class reference and implementation .

Samco Model

The `SamcoFeeModel` models the fees of Samco .

```
security.SetFeeModel(new SamcoFeeModel());
```

The `SamcoFeeModel` models the order fees of Samco by its Equity Intraday fee structure. The following table shows the fees:

Charge Item	Fee
Brokerage Fee	₹20 per trade or 0.02% (whichever is lower)
Exchange Transaction Charge	0.00345%
Securities Transaction Tax	0.025%
Goods and Services Tax	18%
SEBI Charges	0.0001%
Stamp Duty	0.003%

The `SamcoFeeModel` charges fees in INR.

For more information about this model, see the class reference and implementation .

Tastytrade Model

The `TastytradeFeeModel` models the fees of Tastytrade .

```
security.SetFeeModel(new TastytradeFeeModel());
```

The following sections describe the trading fees of each asset class.

For more information about this model, see the class reference and implementation .

US Equities

US Equity trades are free.

Equity Options

Equity Option trades cost $1 USD per contract to open a position, free to close.

Index Options

Index Option trades cost $1 USD per contract to open a position, free to close.

Futures

Futures trades cost $1.25 USD per contract.

Future Options

Future Option trades cost $2.5 USD per contract to open a position, free to close.

TradeStation Model

The `TradeStationFeeModel` models the fees of TradeStation .

```
security.SetFeeModel(new TradeStationFeeModel());
```

The following sections describe the trading fees of each asset class.

For more information about this model, see the class reference and implementation .

US Equities

US Equity trades are free for US residents. For non-US residents, they cost $5 USD per order.

Equity Options

Equity Option trades cost 0.60 USD per contract. For non-US residents, they cost an extra 5 USD per order.

Index Options

Index Option trades cost 1.00 USD per contract. For non-US residents, they cost an extra 5 USD per order.

Futures

Futures trades cost $1.50 USD per contract.

Wolverine Model

The `WolverineFeeModel` models the fees of Wolverine Execution Services.

```
security.SetFeeModel(new WolverineFeeModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`feesPerShare`	`decimal?`	The fees per share to apply. If `null` , it uses 0.005.	`null`

The `WolverineFeeModel` charges fees in USD.

For more information about this model, see the class reference and implementation .

Zerodha Model

The `ZerodhaFeeModel` models the fees of Zerodha .

```
security.SetFeeModel(new ZerodhaFeeModel());
```

The `ZerodhaFeeModel` models the order fees of Zerodha with its Equity Intraday fee structure. The following table shows the fees:

Charge Item	Fee
Brokerage Fee	₹20 per trade or 0.03% (whichever is lower)
Exchange Transaction Charge	0.00345%
Securities Transaction Tax	0.025%
Goods and Services Tax	18%
SEBI Charges	0.0001%
Stamp Duty	0.003%

The `ZerodhaFeeModel` charges fees in INR.

For more information about this model, see the class reference and implementation .

Reality Modeling

Brokerages

Brokerages

Key Concepts

Introduction

Brokerages provide you with a connection to the market so you can fill trades. Brokerage models simulate the live behavior of a real brokerage. To avoid sending invalid orders for execution in live trading, the brokerage model validates your orders before LEAN sends them to the real brokerage. Brokerage models combine together all of the models relevant for a brokerage. If you set the appropriate brokerage model, the supported order types, default markets, and some of the security-level models are appropriately set in your algorithm.

Set Models

To set a brokerage model, in the `Initialize` method, call the `SetBrokerageModel` method with a `BrokerageName` and an `AccountType` . If you set a brokerage model, it overrides any security level models you manually set in your algorithm.

```
C#
public override void Initialize()
{
 // Set the brokerage model to backtest with the most realistic scenario to handle the order validity, margin rate
and transaction fee
 SetBrokerageModel(BrokerageName.OandaBrokerage); // Defaults to margin account
 SetBrokerageModel(BrokerageName.Bitfinex, AccountType.Margin); // Overrides the default account type, which is
AccountType.Cash
}
```

In live trading, LEAN doesn't ignore your `SetBrokerageModel` method calls. LEAN uses some of the brokerage model rules to catch invalid orders before they reach your real brokerage.

To view all the pre-built brokerage models, see Supported Models .

Default Behavior

The default brokerage model is the `DefaultBrokerageModel` , but LEAN has many other brokerage models you can use in your algorithms. For more information about the `DefaultBrokerageModel` , see QuantConnect Paper Trading .

Modeled Properties

Brokerage models model that following properties:

- Transaction fees
- Supported asset classes
- Account types
- Support for extended market hours

- Leverage for assets
- Trade settlement rules
- Order properties and updates

Model Structure

Brokerage models should extend the `DefaultBrokerageModel` class. Extensions of the `DefaultBrokerageModel` class should implement the following methods:

```
C#
public class CustomBrokerageModelExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the custom brokerage model
        SetBrokerageModel(new MyBrokerageModel());
    }
}

// Define the custom brokerage model outside of the algorithm
class MyBrokerageModel : DefaultBrokerageModel
{
    public override decimal RequiredFreeBuyingPowerPercent { get; }

    public new IReadOnlyDictionary<SecurityType, string> DefaultMarkets  = new Dictionary<SecurityType, string>
    {
        {SecurityType.Equity, Market.USA}
    }.ToReadOnlyDictionary();

    public MyBrokerageModel(AccountType accountType = AccountType.Margin)
        : base(accountType)
    {

    }

    public override bool CanSubmitOrder(Security security, Order order,
        out BrokerageMessageEvent message)
    {
        return base.CanSubmitOrder(security, order, out message);
    }

    public override bool CanUpdateOrder(Security security, Order order,
        UpdateOrderRequest request, out BrokerageMessageEvent message)
    {
        return base.CanUpdateOrder(security, order, request, out message);
    }

    public override bool CanExecuteOrder(Security security, Order order)
    {
        return base.CanExecuteOrder(security, order);
    }

    public override void ApplySplit(List<OrderTicket> tickets, Split split)
    {
        base.ApplySplit(tickets, split);
    }

    public override decimal GetLeverage(Security security)
    {
        return base.GetLeverage(security);
    }

    public override IBenchmark GetBenchmark(SecurityManager securities)
    {
        return base.GetBenchmark(securities);
    }

    public override IFillModel GetFillModel(Security security)
    {
        return base.GetFillModel(security);
    }

    public override IFeeModel GetFeeModel(Security security)
    {
        return base.GetFeeModel(security);
    }
```

```
    public override ISlippageModel GetSlippageModel(Security security)
    {
        return base.GetSlippageModel(security);
    }

    public override ISettlementModel GetSettlementModel(Security security)
    {
        return base.GetSettlementModel(security);
    }

    public override IBuyingPowerModel GetBuyingPowerModel(Security security)
    {
        return base.GetBuyingPowerModel(security);
    }

    public override IMarginInterestRateModel GetMarginInterestRateModel(Security security)
    {
        return base.GetMarginInterestRateModel(security);
    }

    public override IShortableProvider GetShortableProvider(Security security)
    {
        return base.GetShortableProvider(security);
    }
}
```

Custom brokerage models give you enormous control over your algorithm behavior and allow you to model virtually any brokerage in the world.

Examples

The following examples demonstrate common practices for implementing a custom brokerage model.

Example 1: Default Model Wrapper

The following algorithm defines a subclass of the `DefaultBrokerageModel` and overrides the built-in members so you can easily customize the parts you need.

C#

```
public class CustomBrokerageModelAlgorithm : QCAlgorithm
{

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        SetBrokerageModel(new MyBrokerageModel());
        AddEquity("SPY", Resolution.Daily);
    }

    /// OnData event is the primary entry point for your algorithm. Each new data point will be pumped in here.
    /// Slice object keyed by symbol containing the stock data
    public override void OnData(Slice data)
    {
        if (!Portfolio.Invested)
        {
            SetHoldings("SPY", 1);
        }
    }

}

class MyBrokerageModel : DefaultBrokerageModel
{
    public override decimal RequiredFreeBuyingPowerPercent { get; }

    public new IReadOnlyDictionary<SecurityType, string> DefaultMarkets  = new Dictionary<SecurityType, string>
    {
        {SecurityType.Equity, Market.USA}
    }.ToReadOnlyDictionary();

    public MyBrokerageModel(AccountType accountType = AccountType.Margin)
```

```
        : base(accountType)
    {

    }

    public override bool CanSubmitOrder(Security security, Order order,
        out BrokerageMessageEvent message)
    {
        return base.CanSubmitOrder(security, order, out message);
    }

    public override bool CanUpdateOrder(Security security, Order order,
        UpdateOrderRequest request, out BrokerageMessageEvent message)
    {
        return base.CanUpdateOrder(security, order, request, out message);
    }

    public override bool CanExecuteOrder(Security security, Order order)
    {
        return base.CanExecuteOrder(security, order);
    }

    public override void ApplySplit(List<OrderTicket> tickets, Split split)
    {
        base.ApplySplit(tickets, split);
    }

    public override decimal GetLeverage(Security security)
    {
        return base.GetLeverage(security);
    }

    public override IBenchmark GetBenchmark(SecurityManager securities)
    {
        return base.GetBenchmark(securities);
    }

    public override IFillModel GetFillModel(Security security)
    {
        return base.GetFillModel(security);
    }

    public override IFeeModel GetFeeModel(Security security)
    {
        return base.GetFeeModel(security);
    }

    public override ISlippageModel GetSlippageModel(Security security)
    {
        return base.GetSlippageModel(security);
    }

    public override ISettlementModel GetSettlementModel(Security security)
    {
        return base.GetSettlementModel(security);
    }

    public override IBuyingPowerModel GetBuyingPowerModel(Security security)
    {
        return base.GetBuyingPowerModel(security);
    }

    public override IMarginInterestRateModel GetMarginInterestRateModel(Security security)
    {
        return base.GetMarginInterestRateModel(security);
    }

    public override IShortableProvider GetShortableProvider(Security security)
    {
        return base.GetShortableProvider(security);
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
BrokerageModelAlgorithm.cs C# CustomBrokerageMessageHandlerAlgorithm.cs C#

Brokerages

Supported Models

Brokerage models combine together all of the models relevant for a brokerage. The following brokerage models are available:

QuantConnect Paper Trading

Interactive Brokers

TradeStation

Tastytrade

Alpaca

Charles Schwab

Binance

Bybit

Tradier

Kraken

Coinbase

Bitfinex

dYdX

Terminal Link

SSC Eze

Trading Technologies

Wolverine

Oanda

See Also

Reality Modeling
Live Trading Brokerages

Supported Models

QuantConnect Paper Trading

Introduction

This page explains the `DefaultBrokerageModel` , including the asset classes it supports, its default security-level models , and its default markets.

```
C#

SetBrokerageModel(BrokerageName.QuantConnectBrokerage, AccountType.Cash);
SetBrokerageModel(BrokerageName.QuantConnectBrokerage, AccountType.Margin); // Overrides the default account type
```

For more information about this model, see the class reference and implementation .

Asset Classes

The `DefaultBrokerageModel` supports trading for all asset classes .

Orders

The following sections describe how the `DefaultBrokerageModel` handles orders.

Order Types

The following table describes the available order types for each asset class that the `DefaultBrokerageModel` supports:

Order Type	US Equity	Equity Options	Crypto	Crypto Futures	Forex	CFD	Futures	Futures Options	Index Options
Market	✓	✓	✓	✓	✓	✓	✓	✓	✓
Limit	✓	✓	✓	✓	✓	✓	✓	✓	✓
Limit if touched	✓	✓	✓	✓	✓	✓	✓	✓	✓
Stop market	✓	✓	✓	✓	✓	✓	✓	✓	✓
Stop limit	✓	✓	✓	✓	✓	✓	✓	✓	✓
Market on open	✓	✓	✓		✓	✓			✓
Market on close	✓	✓	✓		✓	✓	✓	✓	✓
Combo market		✓						✓	✓
Combo limit		✓						✓	✓
Combo leg limit		✓						✓	✓
Exercise Option		✓ Not supported for cash-settled Options						✓	

In live trading, Option orders require a third-party data provider that supports Options. To view the data providers that support Options, see their respective documentation in Cloud Platform , Local Platform , or the CLI .

Time In Force

The `DefaultBrokerageModel` supports the following TimeInForce instructions:

- `Day`
- `GoodTilCanceled`
- `GoodTilDate`

Updates

The `DefaultBrokerageModel` supports order updates .

Handling Splits

If you're using raw data normalization and you have active orders with a limit, stop, or trigger price in the market for a US Equity when a stock split occurs, the following properties of your orders automatically adjust to reflect the stock split:

- Quantity
- Limit price
- Stop price
- Trigger price

Fills

The following table shows the fill model that the `DefaultBrokerageModel` uses for each `SecurityType` :

`SecurityType`	**Fill Model**
`Equity`	EquityFillModel
`Future`	FutureFillModel
`FutureOption`	FutureOptionFillModel
Remaining `SecurityType` values	ImmediateFillModel

Slippage

The `DefaultBrokerageModel` uses the NullSlippageModel .

Fees

The `DefaultBrokerageModel` uses the ConstantFeeModel with no fees for Forex, CFD, Crypto, and Crypto Future assets and the InteractiveBrokersFeeModel for the remaining asset classes.

```
C#
// For Forex, CFD, Crypto, and Crypto Future assets:
security.SetFeeModel(new ConstantFeeModel(0.0m));

// For the remaining asset classes:
security.SetFeeModel(new InteractiveBrokersFeeModel());
```

Buying Power

The `DefaultBrokerageModel` sets the buying power model based on the asset class of the security. The following table shows the default buying power model of each asset class:

Asset Class	Model
Equity Options	`OptionMarginModel`
Futures	`FutureMarginModel`
Future Options	`FuturesOptionsMarginModel`
Index Options	`OptionMarginModel`
Crypto	`CashBuyingPowerModel` for cash accounts or `SecurityMarginModel` for margin accounts
CryptoFuture	`CryptoFutureMarginModel`
Forex	`CashBuyingPowerModel` for cash accounts or `SecurityMarginModel` for margin accounts
Other	`SecurityMarginModel`

If you have a margin account, the `DefaultBrokerageModel` allows 2x leverage for Equities, 50x leverage for Forex and CFDs, and 1x leverage for the remaining asset classes.

Settlement

The following table shows which settlement model the `DefaultBrokerageModel` uses based on the security type and your account type:

Security Type	Account Type	Settlement Model
Equity	Cash	DelayedSettlementModel with the default settlement rules
Option	Cash	DelayedSettlementModel with the default settlement rules
Future	Any	FutureSettlementModel

For all other cases, the `DefaultBrokerageModel` uses the ImmediateSettlementModel .

```C#
// For US Equities with a cash account:
security.SetSettlementModel(new DelayedSettlementModel(Equity.DefaultSettlementDays, Equity.DefaultSettlementTime));

// For Equity Options with a cash account:
security.SetSettlementModel(new DelayedSettlementModel(Option.DefaultSettlementDays, Option.DefaultSettlementTime));

// For Futures
security.SetSettlementModel(new FutureSettlementModel());

// For remaining cases:
security.SetSettlementModel(new ImmediateSettlementModel());
```

Margin Interest Rate

The `DefaultBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The following table describes the default markets of each `SecurityType` for the `DefaultBrokerageModel` :

SecurityType	Market
Equity	USA
Option	USA
Future	CME
FutureOption	CME
Index	USA
IndexOption	USA
Forex	Oanda
Cfd	Oanda
Crypto	Coinbase
CryptoFuture	Binance

Account Currency

The `DefaultBrokerageModel` doesn't set a default currency. To change the algorithm's currency from USD to a different currency, see Set Account Currency .

Supported Models

Interactive Brokers

Introduction

This page explains the `InteractiveBrokersBrokerageModel` , including the asset classes it supports, its default security-level models , and its default markets.

```C#
SetBrokerageModel(BrokerageName.InteractiveBrokersBrokerage, AccountType.Cash);
SetBrokerageModel(BrokerageName.InteractiveBrokersBrokerage, AccountType.Margin);
```

For more information about this model, see the class reference and implementation .

Asset Classes

The `InteractiveBrokersBrokerageModel` supports the following asset classes:

- US Equities
- Equity Options
- Forex
- Futures
- Future Options
- Indices
- Index Options
- CFD

Orders

The `InteractiveBrokersBrokerageModel` supports several order types, order properties, and order updates.

Order Types

The following table describes the order types that the `InteractiveBrokersBrokerageModel` supports: supports. For specific details about each order type, refer to the IB documentation.

Order Type	IB Documentation Page
Market	Market Orders
Limit	Limit Orders
Limit if touched	Limit if Touched Orders
Stop market	Stop Orders
Stop limit	Stop-Limit Orders
Trailing stop	Trailing Stop Orders
Market on open	Market-on-Open (MOO) Orders
Market on close	Market-on-Close (MOC) Orders
Combo market	Spread Orders
Combo limit	Spread Orders
Combo leg limit	Spread Orders
Exercise Option	Options Exercise

The following table describes the available order types for each asset class that the `InteractiveBrokersBrokerageModel` supports:

Order Type	US Equity	Equity Options	Forex	Futures	Futures Options	Index Options	CFD
Market	✓	✓	✓	✓	✓	✓	✓
Limit	✓	✓	✓	✓	✓	✓	✓
Limit if touched	✓	✓	✓	✓	✓	✓	✓
Stop market	✓	✓	✓	✓	✓	✓	✓
Stopl imit	✓	✓	✓	✓	✓	✓	✓
Trailing stop	✓	✓	✓	✓	✓	✓	✓
Market on open	✓	✓			✓	✓	✓
Market on close	✓			✓			✓
Combo market		✓			✓	✓	
Combo limit		✓			✓	✓	
Combo leg limit		✓			✓	✓	
Exercise Option		✓ Not supported for cash-settled Options			✓		

Order Properties

The `InteractiveBrokersBrokerageModel` supports custom order properties. The following table describes the members of the `InteractiveBrokersOrderProperties` object that you can set to customize order execution. The table does not include the methods for FA accounts .

Property	Data Type	Description	Default Value
`TimeInForce`	`TimeInForce`	A TimeInForce instruction to apply to the order. The following instructions are supported: • `Day` • `GoodTilCanceled` • `GoodTilDate`	`TimeInForce.GoodTilCanceled`
`OutsideRegularTradingHours`	`bool`	A flag to signal that the order may be triggered and filled outside of regular trading hours.	`false`

```C#
public override void Initialize()
{
    // Set the default order properties
    DefaultOrderProperties = new InteractiveBrokersOrderProperties
    {
        TimeInForce = TimeInForce.GoodTilCanceled,
        OutsideRegularTradingHours = false
    };
}

public override void OnData(Slice slice)
{
    // Use default order order properties
    LimitOrder(_symbol, quantity, limitPrice);

    // Override the default order properties
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new InteractiveBrokersOrderProperties
               {
                   TimeInForce = TimeInForce.Day,
                   OutsideRegularTradingHours = false
               });
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new InteractiveBrokersOrderProperties
               {
                   TimeInForce = TimeInForce.GoodTilDate(new DateTime(year, month, day)),
                   OutsideRegularTradingHours = true
               });
}
```

Updates

The `InteractiveBrokersBrokerageModel` supports order updates .

Financial Advisor Group Orders

To place FA group orders, see Financial Advisors .

Fractional Trading

The `InteractiveBrokersBrokerageModel` doesn't support fractional trading .

Handling Splits

If you're using raw data normalization and you have active orders with a limit, stop, or trigger price in the market for a US Equity when a stock split occurs, the following properties of your orders automatically adjust to reflect the stock split:

- Quantity

- Limit price
- Stop price
- Trigger price

Order Size Limits

The `InteractiveBrokersBrokerageModel` enforces the Spot Currency Minimum/Maximum Order Sizes from the IB website.

Fills

The following table shows the fill model that the `InteractiveBrokersBrokerageModel` uses for each `SecurityType` :

`SecurityType`	**Fill Model**
`Equity`	EquityFillModel
`Future`	FutureFillModel
`FutureOption`	FutureOptionFillModel
Remaining `SecurityType` values	ImmediateFillModel

Slippage

The `InteractiveBrokersBrokerageModel` uses the NullSlippageModel .

Fees

The `InteractiveBrokersBrokerageModel` uses the InteractiveBrokersFeeModel with the default argument values. We model current Interactive Brokers fees on all assets.

Buying Power

The `InteractiveBrokersBrokerageModel` sets the buying power model based on the asset class of the security. The following table shows the default buying power model of each asset class:

Asset Class	**Model**
Equity Options	`OptionMarginModel`
Futures	`FutureMarginModel`
Future Options	`FuturesOptionsMarginModel`
Index Options	`OptionMarginModel`
Crypto	`CashBuyingPowerModel` for cash accounts or `SecurityMarginModel` for margin accounts
CryptoFuture	`CryptoFutureMarginModel`
Forex	`CashBuyingPowerModel` for cash accounts or `SecurityMarginModel` for margin accounts
Other	`SecurityMarginModel`

If you have a margin account, the `InteractiveBrokersBrokerageModel` allows 2x leverage for Equities, 50x leverage for Forex, 10x leverage for CFDs, and 1x leverage for the remaining asset classes.

Settlement

The `InteractiveBrokersBrokerageModel` uses the ImmediateSettlementModel in most cases. If you trade US Equities or Equity Options with a cash account, it uses the DelayedSettlementModel with the default settlement rules .

The following table shows which settlement model the `InteractiveBrokersBrokerageModel` uses based on the security type and your account type:

Security Type	Account Type	Settlement Model
Equity	Cash	DelayedSettlementModel with the default settlement rules
Option	Cash	DelayedSettlementModel with the default settlement rules
Future	Any	FutureSettlementModel

For all other cases, the `InteractiveBrokersBrokerageModel` uses the ImmediateSettlementModel .

```C#
// For US Equities with a cash account:
security.SetSettlementModel(new DelayedSettlementModel(Equity.DefaultSettlementDays, Equity.DefaultSettlementTime));

// For Equity Options with a cash account:
security.SetSettlementModel(new DelayedSettlementModel(Option.DefaultSettlementDays, Option.DefaultSettlementTime));

// For remaining cases:
security.SetSettlementModel(new ImmediateSettlementModel());
```

Interactive Brokers doesn't provide information on which assets aren't settled, so we assume each live trading session starts with its cash fully settled.

Margin Interest Rate

The `InteractiveBrokersBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The following table describes the default markets of each `SecurityType` for the `InteractiveBrokersBrokerageModel` :

SecurityType	Market
Equity	USA
Option	USA
Future	CME
FutureOption	CME
Index	USA
IndexOption	USA
Forex	Oanda
Cfd	InteractiveBrokers

Financial Advisors

IB supports FA accounts for Trading Firm and Institution organizations. FA accounts enable certified professionals to use a single trading algorithm to manage several client accounts.

To place trades using a subset of client accounts, create Account Groups in Trader Workstation and then define the `InteractiveBrokersOrderProperties` when you create orders.

```
C#
DefaultOrderProperties = new InteractiveBrokersOrderProperties
{
    FaGroup = "TestGroupEQ",
    FaMethod = "Equal",
    Account = "DU123456"
};
```

SecurityHolding objects aggregate your positions across all the account groups. If you have two groups where group A has 10 shares of SPY and group B has -10 shares of SPY, then `Portfolio["SPY"].Quantity` is zero.

LEAN supports several allocation methods for FA group orders. If you intend to use the same group allocation method for every order, set the `DefaultOrderProperties` of your algorithm, which sets the order properties for all of your orders.

```
C#
public override void Initialize()
{
    // Set the default order properties
    DefaultOrderProperties = new InteractiveBrokersOrderProperties()
    {
        FaGroup = "TestGroupEQ",
        FaMethod = "Equal",
        Account = "DU123456"
    };
}

public override void OnData(Slice slice)
{
    // Use default order order properties
    LimitOrder(_symbol, quantity, limitPrice);
}
```

To adjust the order properties of an order, change the `DefaultOrderProperties` or pass an order properties object to the order

method . The following sections explain the FA group allocation methods.

Equal

This group allocation method distributes shares equally between all accounts in the group. When you use this method, you need to specify an order quantity.

For example, say your Account Group includes four accounts and you place an order to buy 400 shares of a stock. In this case, each account receives 100 shares. If your Account Group includes six accounts, each account receives 66 shares, and then 1 share is allocated to each account until all are distributed. After you submit the order, your algorithm receives order events to track the order progress. When all the shares are bought, the order status is `OrderStatus.Filled` . If one of the accounts in the group can't afford 10 of the shares it needs to buy, 10 shares are cancelled and you'll only end up buying 390 shares in total.

```C#
LimitOrder(
    _symbol, quantity, limitPrice,
    orderProperties: new InteractiveBrokersOrderProperties
    {
        FaMethod = "Equal"
    }
);
```

Net Liquidation Value

This group allocation method distributes shares based on the net liquidation value of each account. The system calculates ratios based on the net liquidation value in each account and allocates shares based on these ratios. When you use this method, you need to specify an order quantity.

For example, say your account group includes three accounts, A, B and C with Net Liquidation values of 25,000, 50,000 and $100,000, respectively. In this case, the system calculates a ratio of 1:2:4. If you place an order for 700 shares of a stock, it allocates 100 shares to Client A, 200 shares to Client B, and 400 shares to Client C.

```C#
LimitOrder(_symbol, quantity, limitPrice,
    orderProperties: new InteractiveBrokersOrderProperties
    {
        FaMethod = "NetLiq"
    }
);
```

Available Equity

This group allocation method distributes shares based on the amount of available equity in each account. The system calculates ratios based on the available equity in each account and allocates shares based on these ratios. When you use this method, you need to specify an order quantity.

For example, say your account group includes three accounts, A, B and C with available equity of 25,000, 50,000 and $100,000, respectively. In this case, the system calculates a ratio of 1:2:4. If you place an order for 700 shares of a stock, it allocates 100 shares to Client A, 200 shares to Client B, and 400 shares to Client C.

```C#
LimitOrder(_symbol, quantity, limitPrice,
    orderProperties: new InteractiveBrokersOrderProperties
    {
        FaMethod = "AvailableEquity"
    }
);
```

Account Currency

The `InteractiveBrokersBrokerageModel` doesn't set a default currency. To change the algorithm's currency from USD to a different currency, see Set Account Currency .

Supported Models

TradeStation

Introduction

This page explains the `TradeStationBrokerageModel`, including the asset classes it supports, its default security-level models, and its default markets.

```C#
SetBrokerageModel(BrokerageName.TradeStation, AccountType.Cash);
SetBrokerageModel(BrokerageName.TradeStation, AccountType.Margin);
```

For more information about this model, see the class reference and implementation.

Asset Classes

The `TradeStationBrokerageModel` supports trading the following asset classes:

- US Equities
- Equity Options
- Futures
- Index
- Index Options

Orders

The `TradeStationBrokerageModel` supports several order types, order properties, and order updates.

Order Types

The following table describes the available order types for each asset class that the `TradeStationBrokerageModel` supports:

Order Type	Equity	Equity Options	Index Options	Futures
Market	✓	✓	✓	✓
Limit	✓	✓	✓	✓
Stop market	✓	✓	✓	✓
Stop limit	✓	✓	✓	✓
Trailing stop	✓	✓	✓	✓
Market on Open	✓			
Market on Close	✓			
Combo market	✓	✓	✓	✓
Combo limit	✓	✓	✓	✓

Order Properties

The `TradeStationBrokerageModel` supports custom order properties. The following table describes the members of the `TradeStationOrderProperties` object that you can set to customize order execution.

Property	Data Type	Description	Default Value
`TimeInForce`	`TimeInForce`	A TimeInForce instruction to apply to the order. The following instructions are supported: • `Day` • `GoodTilCanceled` • `GoodTilDate`	`TimeInForce.GoodTilCanceled`
`OutsideRegularTradingHours`	`bool`	A flag to signal that the order may be triggered and filled outside of regular trading hours.	`false`
`PostOnly`	`bool`	This flag will ensure the limit order executes only as a maker (no fee) order. If part of the order results in taking liquidity rather than providing, it will be rejected and no part of the order will execute. Equities only.	`false`

C#

```
public override void Initialize()
{
    // Set the default order properties
    DefaultOrderProperties = new TradeStationOrderProperties
    {
        OutsideRegularTradingHours = false,
        TimeInForce = TimeInForce.GoodTilCanceled
    };
}

public override void OnData(Slice slice)
{
    // Use default order order properties
    LimitOrder(_symbol, quantity, limitPrice);

    // Override the default order properties
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new TradeStationOrderProperties
               {
                   PostOnly = true,
                   TimeInForce = TimeInForce.Day
               });
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new TradeStationOrderProperties
               {
                   OutsideRegularTradingHours = true,
                   TimeInForce = TimeInForce.GoodTilDate(new DateTime(year, month, day))
               });
}
```

Updates

The `TradeStationBrokerageModel` supports order updates .

Handling Splits

If you're using raw data normalization and you have active orders with a limit, stop, or trigger price in the market for a US Equity when a stock split occurs, the following properties of your orders automatically adjust to reflect the stock split:

- Quantity
- Limit price
- Stop price
- Trigger price

Fills

The following table shows the fill model that the `TradeStationBrokerageModel` uses for each `SecurityType` :

`SecurityType`	**Fill Model**
`Equity`	EquityFillModel
`Option`	ImmediateFillModel
`Future`	FutureFillModel

Slippage

The `TradeStationBrokerageModel` uses the NullSlippageModel .

Fees

The `TradeStationBrokersBrokerageModel` uses the TradeStationFeeModel with the default argument values. We model current TradeStation fees on all assets.

Buying Power

The `TradeStationBrokerageModel` sets the buying power model based on the asset class of the security. The following table shows the default buying power model of each asset class:

Asset Class	**Model**
Equities	`SecurityMarginModel`
Equity Options	`OptionMarginModel`
Futures	`FutureMarginModel`

If you have a margin account, the `TradeStationBrokerageModel` allows 2x leverage for Equities and and 1x leverage for Equity Options and Futures.

Settlement

The following table shows which settlement model the `TradeStationBrokerageModel` uses based on the security type and your account type:

Security Type	Account Type	Settlement Model
Equity	Cash	DelayedSettlementModel with the default settlement rules
Option	Cash	DelayedSettlementModel with the default settlement rules
Future	Any	FutureSettlementModel

For all other cases, the `TradeStationBrokerageModel` uses the ImmediateSettlementModel .

```C#
// For US Equities with a cash account:
security.SetSettlementModel(new DelayedSettlementModel(Equity.DefaultSettlementDays, Equity.DefaultSettlementTime));

// For Equity Options with a cash account:
security.SetSettlementModel(new DelayedSettlementModel(Option.DefaultSettlementDays, Option.DefaultSettlementTime));

// For Futures
security.SetSettlementModel(new FutureSettlementModel());

// For remaining cases:
security.SetSettlementModel(new ImmediateSettlementModel());
```

Margin Interest Rate

The `TradeStationBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The following table describes the default markets of each `SecurityType` for the `TradeStationBrokerageModel` :

SecurityType	Market
Equity	USA
Option	USA
Future	CME

Account Currency

The `TradeStationBrokerageModel` doesn't set a default currency. To change the algorithm's currency from USD to a different currency, see Set Account Currency .

Supported Models

Tastytrade

Introduction

This page explains the `TastytradeBrokerageModel` , including the asset classes it supports, its default security-level models , and its default markets.

```
SetBrokerageModel(BrokerageName.Tastytrade, AccountType.Cash);
SetBrokerageModel(BrokerageName.Tastytrade, AccountType.Margin);
```

For more information about this model, see the class reference and implementation .

Asset Classes

The `TastytradeBrokerageModel` supports the following asset classes:

- US Equities
- Equity Options
- Futures
- Future Options
- Indices
- Index Options

Orders

The `TastytradeBrokerageModel` supports several order types, order properties, and order updates.

Order Types

The following table describes the available order types for each asset class that the `TastytradeBrokerageModel` supports:

Order Type	Equity	Equity Options	Futures	Future Options	Index Options
Market	✓	✓	✓	✓	✓
Limit	✓	✓	✓	✓	✓
Stop market	✓	✓	✓	✓	✓
Stop limit	✓	✓	✓	✓	✓
Combo limit	✓	✓	✓	✓	✓

Order Properties

The `TastytradeBrokerageModel` supports custom order properties. The following table describes the members of the `TastytradeOrderProperties` object that you can set to customize order execution.

Property	Data Type	Description	Default Value
`TimeInForce`	`TimeInForce`	A TimeInForce instruction to apply to the order. The following instructions are supported: • `Day` • `GoodTilCanceled` • `GoodTilDate`	`TimeInForce.GoodTilCanceled`

```
C#
public override void Initialize()
{
    // Set the default order properties
    DefaultOrderProperties = new TastytradeOrderProperties
    {
        TimeInForce = TimeInForce.GoodTilCanceled
    };
}

public override void OnData(Slice slice)
{
    // Use default order order properties
    LimitOrder(_symbol, quantity, limitPrice);

    // Override the default order properties
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new TastytradeOrderProperties
               {
                   TimeInForce = TimeInForce.Day
               });
}
```

Updates

The `TastytradeBrokerageModel` supports order updates .

Handling Splits

If you're using raw data normalization and you have active orders with a limit, stop, or trigger price in the market for a US Equity when a stock split occurs, the following properties of your orders automatically adjust to reflect the stock split:

- Quantity
- Limit price
- Stop price
- Trigger price

Fills

The following table shows the fill model that the `TastytradeBrokerageModel` uses for each `SecurityType` :

SecurityType	Fill Model
Equity	EquityFillModel
Future	FutureFillModel
FutureOption	FutureOptionFillModel
Remaining SecurityType values	ImmediateFillModel

Slippage

The TastytradeBrokerageModel uses the NullSlippageModel .

Fees

The TastytradeBrokersBrokerageModel uses the TastytradeFeeModel with the default argument values. We model current Tastytrade fees on all assets.

Buying Power

The TastytradeBrokerageModel sets the buying power model based on the asset class of the security. The following table shows the default buying power model of each asset class:

Asset Class	Model
Equities	SecurityMarginModel
Futures	FutureMarginModel
All Options Types	OptionMarginModel

If you have a margin account, the TastytradeBrokerageModel allows 2x leverage for Equities. The remaining asset classes are derivatives.

Settlement

The following table shows which settlement model the TastytradeBrokerageModel uses based on the security type and your account type:

Security Type	Account Type	Settlement Model
Equity	Cash	DelayedSettlementModel with the default settlement rules
Option	Cash	DelayedSettlementModel with the default settlement rules
Future	Any	FutureSettlementModel

For all other cases, the TastytradeBrokerageModel uses the ImmediateSettlementModel .

```C#
// For US Equities with a cash account:
security.SetSettlementModel(new DelayedSettlementModel(Equity.DefaultSettlementDays, Equity.DefaultSettlementTime));

// For Equity Options with a cash account:
security.SetSettlementModel(new DelayedSettlementModel(Option.DefaultSettlementDays, Option.DefaultSettlementTime));

// For Futures
security.SetSettlementModel(new FutureSettlementModel());

// For remaining cases:
security.SetSettlementModel(new ImmediateSettlementModel());
```

Margin Interest Rate

The `TastytradeBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The following table describes the default markets of each `SecurityType` for the `TastytradeBrokerageModel` :

SecurityType	Market
Equity	USA
Option	USA
Future	CME
FutureOption	CME
Index	USA
IndexOption	USA

Supported Models

Alpaca

Introduction

This page explains the `AlpacaBrokerageModel` , including the asset classes it supports, its default security-level models , and its default markets.

```
SetBrokerageModel(BrokerageName.Alpaca, AccountType.Cash);
SetBrokerageModel(BrokerageName.Alpaca, AccountType.Margin);
```

For more information about this model, see the class reference and implementation .

Asset Classes

The `AlpacaBrokerageModel` supports the following asset classes:

- US Equities
- Equity Options
- Crypto

Orders

The `AlpacaBrokerageModel` supports several order types, order properties, and most order updates.

Order Types

The following table describes the available order types for each asset class that the `AlpacaBrokerageModel` supports:

Order Type	Equity	Equity Options	Crypto
Market	✓	✓	✓
Limit	✓	✓	✓
Stop market	✓	✓	✓
Stop limit	✓	✓	✓
Market on Open	✓		
Market on Close	✓		

Order Properties

The `AlpacaBrokerageModel` supports custom order properties. The following table describes the members of the `AlpacaOrderProperties` object that you can set to customize order execution.

Property	Data Type	Description	Default Value
`TimeInForce`	`TimeInForce`	A TimeInForce instruction to apply to the order. The following instructions are supported: • `Day` • `GoodTilCanceled` • `GoodTilDate`	`TimeInForce.GoodTilCanceled`
`OutsideRegularTradingHours`	`bool`	A flag to signal that the order may be triggered and filled outside of regular trading hours.	`false`

```
C#
public override void Initialize()
{
    // Set the default order properties
    DefaultOrderProperties = new AlpacaOrderProperties
    {
        OutsideRegularTradingHours = false,
        TimeInForce = TimeInForce.GoodTilCanceled
    };
}

public override void OnData(Slice slice)
{
    // Use default order order properties
    LimitOrder(_symbol, quantity, limitPrice);

    // Override the default order properties
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new AlpacaOrderProperties
               {
                   TimeInForce = TimeInForce.Day
               });
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new AlpacaOrderProperties
               {
                   OutsideRegularTradingHours = true,
                   TimeInForce = TimeInForce.GoodTilDate(new DateTime(year, month, day))
               });
}
```

Updates

The `AlpacaBrokerageModel` supports order updates .

Handling Splits

If you're using raw data normalization and you have active orders with a limit, stop, or trigger price in the market for a US Equity when a stock split occurs, the following properties of your orders automatically adjust to reflect the stock split:

- Quantity
- Limit price
- Stop price
- Trigger price

Fills

The `AlpacaBrokerageModel` uses the EquityFillModel for Equity trades and the ImmediateFillModel for Option and Crypto trades.

Slippage

The `AlpacaBrokerageModel` uses the NullSlippageModel .

Fees

The `AlpacaBrokerageModel` uses the AlpacaFeeModel with the default argument values. We model current Alpaca fees on all assets.

Buying Power

The `AlpacaBrokerageModel` uses the `CashBuyingPowerModel` for cash accounts, the `SecurityMarginModel` for margin accounts, and the `OptionMarginModel` for Option trades.

If you have a margin account, the `AlpacaBrokerageModel` allows 2x leverage for Equities and 3x leverage for Crypto.

Settlement

The `AlpacaBrokerageModel` uses the ImmediateSettlementModel for margin accounts and the DelayedSettlementModel with the default settlement rules for cash accounts with Equity and Equity Options.

```C#
// For US Equities with a cash account:
security.SetSettlementModel(new DelayedSettlementModel(Equity.DefaultSettlementDays, Equity.DefaultSettlementTime));

// For Equity Options with a cash account:
security.SetSettlementModel(new DelayedSettlementModel(Option.DefaultSettlementDays, Option.DefaultSettlementTime));

// For remaining cases:
security.SetSettlementModel(new ImmediateSettlementModel());
```

Margin Interest Rate

The `AlpacaBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The default market of the `AlpacaBrokerageModel` is `Market.USA` for Equity and Equity Options. It's `Market.Coinbase` for Crypto.

Account Currency

The `AlpacaBrokerageModel` doesn't set a default currency. To change the algorithm's currency from USD to a different currency, see Set Account Currency .

Supported Models

Charles Schwab

Introduction

This page explains the `CharlesSchwabBrokerageModel`, including the asset classes it supports, its default security-level models, and its default markets.

```C#
SetBrokerageModel(BrokerageName.CharlesSchwab, AccountType.Cash);
SetBrokerageModel(BrokerageName.CharlesSchwab, AccountType.Margin);
```

For more information about this model, see the class reference and implementation.

Asset Classes

The `CharlesSchwabBrokerageModel` supports the following asset classes:

- US Equities
- Equity Options
- Index
- Index Options

Orders

The `CharlesSchwabBrokerageModel` supports several order types, order properties, and order updates.

Order Types

The following table describes the available order types for each asset class that the `CharlesSchwabBrokerageModel` supports:

Order Type	Equity	Equity Options	Index Options
Market	✓	✓	✓
Limit	✓	✓	✓
Stop market	✓	✓	
Market on open	✓		
Market on close	✓		
Combo market	✓	✓	✓
Combo limit	✓	✓	✓

Order Properties

The `CharlesSchwabBrokerageModel` supports custom order properties. The following table describes the members of the `CharlesSchwabOrderProperties` object that you can set to customize order execution.

Property	Data Type	Description	Default Value
`TimeInForce`	`TimeInForce`	A TimeInForce instruction to apply to the order. The following instructions are supported: • `Day` • `GoodTilCanceled` • `GoodTilDate`	`TimeInForce.GoodTilCanceled`
`ExtendedRegularTradingHours`	`bool`	If set to true, allows orders to also trigger or fill outside of regular trading hours.	`false`

```C#
public override void Initialize()
{
    // Set the default order properties
    DefaultOrderProperties.TimeInForce = TimeInForce.GoodTilCanceled;
}

public override void OnData(Slice slice)
{
    // Use default order order properties
    LimitOrder(_symbol, quantity, limitPrice);

    // Override the default order properties
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new CharlesSchwabOrderProperties
               {
                   TimeInForce = TimeInForce.Day,
                   ExtendedRegularTradingHours = false
               });
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new CharlesSchwabOrderProperties
               {
                   TimeInForce = TimeInForce.GoodTilDate(new DateTime(year, month, day))
               });
}
```

Updates

The `CharlesSchwabBrokerageModel` supports order updates .

Handling Splits

If you're using raw data normalization and you have active orders with a limit, stop, or trigger price in the market for a US Equity when a stock split occurs, the following properties of your orders automatically adjust to reflect the stock split:

- Quantity
- Limit price
- Stop price
- Trigger price

Fills

The `CharlesSchwabBrokerageModel` uses the EquityFillModel for Equity trades and the ImmediateFillModel for Equity and Index Options trades.

Slippage

The `CharlesSchwabBrokerageModel` uses the NullSlippageModel .

Fees

The `CharlesSchwabBrokerageModel` uses the CharlesSchwabFeeModel .

Buying Power

The `CharlesSchwabBrokerageModel` uses the `SecurityMarginModel` . If you have a margin account, the `CharlesSchwabBrokerageModel` allows up to 2x leverage.

Settlement

The `CharlesSchwabBrokerageModel` uses the ImmediateSettlementModel for margin accounts and the DelayedSettlementModel with the default settlement rules for cash accounts.

```
C#
// For cash accounts:
security.SetSettlementModel(new DelayedSettlementModel(Equity.DefaultSettlementDays, Equity.DefaultSettlementTime));

// For margin accounts:
security.SetSettlementModel(new ImmediateSettlementModel());
```

Margin Interest Rate

The `CharlesSchwabBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The default market of the `CharlesSchwabBrokerageModel` is `Market.USA` .

Account Currency

The `CharlesSchwabBrokerageModel` doesn't set a default currency. To change the algorithm's currency from USD to a different currency, see Set Account Currency .

Supported Models

Binance

Introduction

This page explains the Binance brokerage models, including the asset classes they supports, their default security-level models , and their default markets.

```
C#

// Binance Spot
SetBrokerageModel(BrokerageName.Binance, AccountType.Cash);
SetBrokerageModel(BrokerageName.Binance, AccountType.Margin);

// Binance Futures
SetBrokerageModel(BrokerageName.BinanceFutures, AccountType.Margin);
SetBrokerageModel(BrokerageName.BinanceCoinFutures, AccountType.Margin);

// Binance US Spot
SetBrokerageModel(BrokerageName.BinanceUS, AccountType.Cash);
```

For more information about these models, see their respective class reference and implementation.

Model	Class Reference	Implementation
BinanceBrokerageModel		
BinanceFuturesBrokerageModel		
BinanceCoinFuturesBrokerageModel		
BinanceUSBrokerageModel		

Asset Classes

The Binance brokerage models support trading Crypto and Crypto Futures .

```
C#

AddCrypto("BTCUSDT", Resolution.Minute, Market.Binance);
AddCryptoFuture("BTCUSD", Resolution.Minute, Market.Binance);
AddCrypto("BTCUSDT", Resolution.Minute, Market.BinanceUS);
```

The Binance US brokerage model supports trading Crypto .

```
C#

AddCrypto("BTCUSDT", Resolution.Minute, Market.BinanceUS);
```

Orders

The Binance and Binance US brokerage models support several order types and order properties, but don't support order updates.

Order Types

The following table describes the available order types for each asset class that the Binance and Binance US brokerage models support:

Order Type	Crypto	Crypto Futures
Market	✓	✓
Limit	✓	✓
Stop Market		✓
Stop limit	✓	✓

Order Properties

The Binance and Binance US brokerage models supports custom order properties. The following table describes the members of the `BinanceOrderProperties` object that you can set to customize order execution:

Property	Data Type	Description	Default Value
`TimeInForce`	`TimeInForce`	A TimeInForce instruction to apply to the order. The following instructions are supported: • `Day` • `GoodTilCanceled` • `GoodTilDate`	`TimeInForce.GoodTilCanceled`
`PostOnly`	`bool`	A flag to signal that the order must only add liquidity to the order book and not take liquidity from the order book. If part of the order results in taking liquidity rather than providing liquidity, the order is rejected without any part of it being filled.	

```C#
public override void Initialize()
{
    // Set the default order properties
    DefaultOrderProperties = new BinanceOrderProperties
    {
        TimeInForce = TimeInForce.GoodTilCanceled,
        PostOnly = false
    };
}

public override void OnData(Slice slice)
{
    // Use default order order properties
    LimitOrder(_symbol, quantity, limitPrice);

    // Override the default order properties
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new BinanceOrderProperties
               {
                   TimeInForce = TimeInForce.Day,
                   PostOnly = false
               });
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new BinanceOrderProperties
               {
                   TimeInForce = TimeInForce.GoodTilDate(new DateTime(year, month, day)),
                   PostOnly = true
               });
}
```

Updates

The Binance and Binance US brokerage models don't support order updates, but you can cancel an existing order and then create a new order with the desired arguments. For more information about this workaround, see the Workaround for Brokerages That Don't Support Updates .

Fills

The Binance brokerage models use the ImmediateFillModel .

Slippage

The Binance brokerage models use the NullSlippageModel .

Fees

The following table shows the fee model that each Binance brokerage model uses:

Brokerage Model	Fee Model
`BinanceBrokerageModel`	BinanceFeeModel
`BinanceFuturesBrokerageModel`	BinanceFuturesFeeModel
`BinanceCoinFuturesBrokerageModel`	BinanceCoinFuturesFeeModel
`BinanceUSBrokerageModel`	BinanceFeeModel

The Binance brokerage models use the default argument values for each fee model so that it models the current Binance fee schedule.

Buying Power

The Binance brokerage models use the `CashBuyingPowerModel` for cash accounts and the `SecurityMarginModel` for margin accounts.

The following table shows the maximum leverage each brokerage model allows for margin accounts:

Brokerage Model	Maximum Leverage
`BinanceBrokerageModel`	3
`BinanceFuturesBrokerageModel`	25
`BinanceCoinFuturesBrokerageModel`	25

The `BinanceUSBrokerageModel` doesn't currently support margin trading.

Settlement

The Binance brokerage models use the ImmediateSettlementModel .

Margin Interest Rate

The following table shows the margin interest rate model that the Binance brokerages use:

Brokerage Model	Margin Interest Rate Model
`BinanceBrokerageModel`	NullMarginInterestRateModel
`BinanceFuturesBrokerageModel`	BinanceFutureMarginInterestRateModel for Crypto Perpetual Futures and NullMarginInterestRateModel for other assets
`BinanceCoinFuturesBrokerageModel`	BinanceFutureMarginInterestRateModel for Crypto Perpetual Futures and NullMarginInterestRateModel for other assets
`BinanceUSBrokerageModel`	NullMarginInterestRateModel

Default Markets

The following table shows the default market of the Binance brokerage models:

Brokerage Model	Market
`BinanceBrokerageModel`	`Market.Binance`
`BinanceFuturesBrokerageModel`	`Market.Binance`
`BinanceCoinFuturesBrokerageModel`	`Market.Binance`
`BinanceUSBrokerageModel`	`Market.BinanceUS`

Account Currency

Most of the Binance brokerage models support fiat currencies like USD, but the `BinanceBrokerageModel` doesn't. When you use this brokerage model, set the account currency to USDT or another Cryptocurrency that Binance supports .

Supported Models

Bybit

Introduction

This page explains the `BybitBrokerageModel` , including the asset classes it supports, its default security-level models , and its default markets.

```C#
SetBrokerageModel(BrokerageName.Bybit, AccountType.Cash);
SetBrokerageModel(BrokerageName.Bybit, AccountType.Margin);
```

For more information about this model, see the class reference and implementation .

Asset Classes

The `BybitBrokerageModel` supports trading Crypto and Crypto Futures .

Orders

The `BybitBrokerageModel` supports several order types, order properties, and order updates.

Order Types

The following table describes the available order types for each asset class that the `BybitBrokerageModel` supports:

Order Type	Crypto	Crypto Futures
Market	✓	✓
Limit	✓	✓
Stop market	✓	✓
Stop limit	✓	✓

Order Properties

The `BybitBrokerageModel` supports custom order properties. The following table describes the members of the `BybitBrokerageModel` object that you can set to customize order execution:

Property	Data Type	Description	Default Value
`TimeInForce`	`TimeInForce`	A TimeInForce instruction to apply to the order. The following instructions are supported: • `Day` • `GoodTilCanceled` • `GoodTilDate`	`TimeInForce.GoodTilCanceled`
`PostOnly`	`bool`	A flag to signal that the order must only add liquidity to the order book and not take liquidity from the order book. If part of the order results in taking liquidity rather than providing liquidity, the order is rejected without any part of it being filled. This order property is only available for limit orders.	
`ReduceOnly`	`bool?`	A flag to signal that the order must only reduce your current position size. For more information about this order property, see Reduce-Only Order on the Bybit website.	

C#

```
public override void Initialize()
{
    // Set the default order properties
    DefaultOrderProperties = new BybitOrderProperties
    {
        TimeInForce = TimeInForce.GoodTilCanceled,
        PostOnly = false,
        ReduceOnly = false
    };
}

public override void OnData(Slice slice)
{
    // Use default order order properties
    LimitOrder(_symbol, quantity, limitPrice);

    // Override the default order properties
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new BybitOrderProperties
               {
                   TimeInForce = TimeInForce.Day,
                   PostOnly = true,
                   ReduceOnly = false
               });
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new BybitOrderProperties
               {
                   TimeInForce = TimeInForce.GoodTilDate(new DateTime(year, month, day)),
                   PostOnly = false,
                   ReduceOnly = true
               });
}
```

Updates

The BybitBrokerageModel supports order updates for Crypto Future assets that have one of the following order states :

- OrderStatus.New
- OrderStatus.PartiallyFilled
- OrderStatus.Submitted
- OrderStatus.UpdateSubmitted

In cases where you can't update an order, you can cancel the existing order and then create a new order with the desired arguments. For more information about this workaround, see the Workaround for Brokerages That Don't Support Updates .

Fills

The BybitBrokerageModel uses the ImmediateFillModel .

Slippage

The BybitBrokerageModel uses the NullSlippageModel .

Fees

The BybitBrokerageModel uses the BybitFeeModel for Crypto trades and the BybitFuturesFeeModel for Crypto Future trades. In both cases, the brokerage model uses the default argument values that models the Bybit fee schedule for VIP Level 0.

Buying Power

The BybitBrokerageModel sets the buying power model based on the asset class of the security. The following table shows the default buying power model of each asset class:

Asset Class	Model
Crypto	CashBuyingPowerModel for cash accounts or SecurityMarginModel for margin accounts
CryptoFuture	CryptoFutureMarginModel

If you have a margin account, the BybitBrokerageModel allows 10x leverage.

Settlement

The BybitBrokerageModel uses the ImmediateSettlementModel .

Margin Interest Rate

The BybitBrokerageModel uses the BybitFutureMarginInterestRateModel for perpetual Futures and the NullMarginInterestRateModel for every other asset.

Default Markets

The default market of the BybitBrokerageModel is Market.Bybit .

Account Currency

The BybitBrokerageModel doesn't set a default currency. To change the algorithm's currency from USD to a different currency, see Set Account Currency .

Supported Models

Tradier

Introduction

This page explains the `TradierBrokerageModel` , including the asset classes it supports, its default security-level models , and its default markets.

```C#
SetBrokerageModel(BrokerageName.TradierBrokerage, AccountType.Cash);
SetBrokerageModel(BrokerageName.TradierBrokerage, AccountType.Margin);
```

For more information about this model, see the class reference and implementation .

Asset Classes

The `TradierBrokerageModel` supports trading US Equities and Equity Options .

Orders

The `TradierBrokerageModel` supports several order types, order properties, and most order updates.

Order Types

The following table describes the available order types for each asset class that the `TradierBrokerageModel` supports:

Order Type	Equity	Equity Options
Market	✓	✓
Limit	✓	✓
Stop market	✓	✓
Stop limit	✓	✓

Order Properties

The `TradierBrokerageModel` supports custom order properties. The following table describes the members of the `TradierOrderProperties` object that you can set to customize order execution.

Property	Data Type	Description	Default Value
`TimeInForce`	`TimeInForce`	A TimeInForce instruction to apply to the order. The following instructions are supported: • `Day` • `GoodTilCanceled` • `GoodTilDate`	`TimeInForce.GoodTilCanceled`
`OutsideRegularTradingHours`	`bool`	A flag to signal that the order may be triggered and filled outside of regular trading hours.	`false`

```C#
public override void Initialize()
{
    // Set the default order properties
    DefaultOrderProperties = new TradierOrderProperties
    {
        OutsideRegularTradingHours = false,
        TimeInForce = TimeInForce.GoodTilCanceled
    };
}

public override void OnData(Slice slice)
{
    // Use default order order properties
    LimitOrder(_symbol, quantity, limitPrice);

    // Override the default order properties
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new TradierOrderProperties
               {
                   TimeInForce = TimeInForce.Day
               });
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new TradierOrderProperties
               {
                   OutsideRegularTradingHours = true,
                   TimeInForce = TimeInForce.GoodTilDate(new DateTime(year, month, day))
               });
}
```

Updates

The `TradierBrokerageModel` supports most order updates . To update the quantity of an order, cancel the order and then submit a new order with the desired quantity. For more information about this workaround, see the Workaround for Brokerages That Don't Support Updates .

Extended Market Hours

The `TradierBrokerageModel` doesn't support extended market hours trading. If you place an order outside of regular trading hours, the order will be processed at market open.

Automatic Cancellations

If you have open orders for a security when it performs a reverse split, the `TradierBrokerageModel` automatically cancels your orders.

Errors

To view the order-related error codes from Tradier, see Error Responses in their documentation.

The `TradierBrokerageModel` validates your orders for the following errors before sending them to Tradier:

Error	Description
`ShortOrderIsGtc`	You can't short sell with the `GoodTilCanceled` time in force
`SellShortOrderLastPriceBelow5`	You can't short sell stocks below $5
`IncorrectOrderQuantity`	The order quantity must be between 1 and 10,000,000 shares

Fills

The `TradierBrokerageModel` uses the EquityFillModel for Equity trades and the ImmediateFillModel for Option trades.

Slippage

The `TradierBrokerageModel` uses the NullSlippageModel .

Fees

The `TradierBrokerageModel` uses the ConstantFeeModel with zero fees.

```
C#
security.SetFeeModel(new ConstantFeeModel(0.0m));
```

Buying Power

The `TradierBrokerageModel` uses the `OptionMarginModel` for Option trades and the `SecurityMarginModel` for Equity trades.

If you have a margin account, the `TradierBrokerageModel` allows 2x leverage for Equities.

Settlement

The `TradierBrokerageModel` uses the ImmediateSettlementModel for margin accounts and the DelayedSettlementModel with the default settlement rules for cash accounts.

```
C#
// For US Equities with a cash account:
security.SetSettlementModel(new DelayedSettlementModel(Equity.DefaultSettlementDays, Equity.DefaultSettlementTime));

// For Equity Options with a cash account:
security.SetSettlementModel(new DelayedSettlementModel(Option.DefaultSettlementDays, Option.DefaultSettlementTime));

// For remaining cases:
security.SetSettlementModel(new ImmediateSettlementModel());
```

Margin Interest Rate

The `TradierBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The default market of the `TradierBrokerageModel` is `Market.USA` .

Account Currency

The `TradierBrokerageModel` doesn't set a default currency. To change the algorithm's currency from USD to a different currency, see Set Account Currency .

Supported Models

Kraken

Introduction

This page explains the `KrakenBrokerageModel` , including the asset classes it supports, its default security-level models , and its default markets.

```
C#
SetBrokerageModel(BrokerageName.Kraken, AccountType.Cash);
SetBrokerageModel(BrokerageName.Kraken, AccountType.Margin);
```

For more information about this model, see the class reference and implementation .

Asset Classes

The `KrakenBrokerageModel` supports trading Crypto .

Orders

The `KrakenBrokerageModel` supports several order types and order properties, but it doesn't support order updates.

Order Types

The following table describes the available order types for each asset class that the `KrakenBrokerageModel` supports:

Order Type	Crypto
Market	✓
Limit	✓
Limit if touched	✓
Stop market	✓
Stop limit	✓

Order Properties

The `KrakenBrokerageModel` supports custom order properties. The following table describes the members of the `KrakenOrderProperties` object that you can set to customize order execution:

Property	Data Type	Description	Default Value
`TimeInForce`	`TimeInForce`	A TimeInForce instruction to apply to the order. The following instructions are supported: • `Day` • `GoodTilCanceled` • `GoodTilDate`	`TimeInForce.GoodTilCanceled`
`PostOnly`	`bool`	A flag to signal that the order must only add liquidity to the order book and not take liquidity from the order book. If part of the order results in taking liquidity rather than providing liquidity, the order is rejected without any part of it being filled.	
`FeeInBase`	`bool`	A flag to signal that the order fees should be paid in the base currency, which is the default behavior when selling. This flag must be the opposite of the `FeeInQuote` flag.	
`FeeInQuote`	`bool`	A flag to signal that the order fees should be paid in the quote currency, which is the default behavior when buying. This flag must be the opposite of the `FeeInBase` flag.	
`NoMarketPriceProtection`	`bool`	A flag to signal that no Market Price Protection should be used.	
`ConditionalOrder`	`Order`	An `Order` that's submitted when the primary order is executed. The `ConditionalOrder` quantity must match the primary order quantity and the `ConditionalOrder` direction must be the opposite of the primary order direction. This order property is only available for live algorithms.	

```C#
public override void Initialize()
{
    // Set the default order properties
    DefaultOrderProperties = new KrakenOrderProperties
    {
        TimeInForce = TimeInForce.GoodTilCanceled,
        PostOnly = false,
        FeeInBase = true,
        FeeInQuote = false,
        NoMarketPriceProtection = true
    };
}

public override void OnData(Slice slice)
{
    // Use default order order properties
    LimitOrder(_symbol, quantity, limitPrice);

    // Override the default order properties
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new KrakenOrderProperties
               {
                   TimeInForce = TimeInForce.Day,
                   PostOnly = true,
                   FeeInBase = false,
                   FeeInQuote = true,
                   NoMarketPriceProtection = true
               });
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new KrakenOrderProperties
               {
                   TimeInForce = TimeInForce.GoodTilDate(new DateTime(year, month, day)),
                   PostOnly = false,
                   FeeInBase = true,
                   FeeInQuote = false,
                   NoMarketPriceProtection = false,
                   ConditionalOrder = StopLimitOrder(_symbol, -quantity, stopLimitPrice, stopPrice)
               });
}
```

Updates

The `KrakenBrokerageModel` doesn't support order updates, but you can cancel an existing order and then create a new order with the desired arguments. For more information about this workaround, see the Workaround for Brokerages That Don't Support Updates .

Fills

The `KrakenBrokerageModel` uses the ImmediateFillModel .

Slippage

The `KrakenBrokerageModel` uses the NullSlippageModel .

Fees

The `KrakenBrokerageModel` uses the KrakenFeeModel .

Buying Power

The `KrakenBrokerageModel` uses the `CashBuyingPowerModel` for cash accounts and the `SecurityMarginModel` for margin accounts.

If you have a margin account, the `KrakenBrokerageModel` allows 1x leverage for most Crypto pairs. The following table shows pairs that have additional leverage available:

Quote Currency	Base Currencies	Leverage
ADA	BTC, ETH, USD, EUR	3
BCH	BTC, USD, EUR	2
BTC	USD, EUR	5
DASH	BTC, USD, EUR	3
EOS	BTC, ETH, USD, EUR	3
ETH	BTC, USD, EUR	5
LINK	BTC, ETH, USD, EUR	3
LTC	BTC, USD, EUR	3
REP	BTC, ETH, USD, EUR	2
TRX	BTC, ETH, USD, EUR	3
USDC	USD, EUR	3
USDT	USD, EUR	2
XMR	BTC, USD, EUR	2
XRP	BTC, USD, EUR	3
XTZ	BTC, ETH, USD, EUR	2

Settlement

The `KrakenBrokerageModel` uses the ImmediateSettlementModel .

Margin Interest Rate

The `KrakenBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The default market of the `KrakenBrokerageModel` is `Market.Kraken` .

Account Currency

The `KrakenBrokerageModel` doesn't set a default currency. To change the algorithm's currency from USD to a different currency, see Set Account Currency .

Supported Models

Coinbase

Introduction

This page explains the `CoinbaseBrokerageModel` , including the asset classes it supports, its default security-level models , and its default markets.

```C#
SetBrokerageModel(BrokerageName.Coinbase, AccountType.Cash);
```

For more information about this model, see the class reference and implementation .

Asset Classes

The `CoinbaseBrokerageModel` supports trading Crypto .

Orders

The `CoinbaseBrokerageModel` supports several order types and order properties, but it doesn't support order updates.

Order Types

The following table describes the available order types for each asset class that the `CoinbaseBrokerageModel` supports:

Order Type	Crypto
Market	✓
Limit	✓
Stop market	✓ Supported before 2019-03-23 in backtests. For reference, see the Coinbase Market Structure Update on the Coinbase website.
Stop limit	✓

Order Properties

The `CoinbaseBrokerageModel` supports custom order properties. The following table describes the members of the `CoinbaseOrderProperties` object that you can set to customize order execution:

Property	Data Type	Description	Default Value
`TimeInForce`	`TimeInForce`	A TimeInForce instruction to apply to the order. The `GoodTilCanceled` `GOOD_TIL_CANCELED` `TimeInForce` is supported.	`TimeInForce.GoodTilCanceled`
`PostOnly`	`bool`	A flag that signals the order must only add liquidity to the order book and not take liquidity from the order book. If part of the order results in taking liquidity rather than providing liquidity, the order is rejected without any part of it being filled.	
`SelfTradePreventionId`	`bool`	A flag that signals self-trade prevention is enabled for this order. Self-trade prevention helps prevent an order from crossing against the same user, reducing the risk of unintentional trades within the same account.	

```C#
public override void Initialize()
{
    // Set the default order properties
    DefaultOrderProperties = new CoinbaseOrderProperties
    {
        TimeInForce = TimeInForce.GoodTilCanceled,
        PostOnly = false
    };
}

public override void OnData(Slice slice)
{
    // Use default order order properties
    LimitOrder(_symbol, quantity, limitPrice);

    // Override the default order properties
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new CoinbaseOrderProperties
               {
                   TimeInForce = TimeInForce.GoodTilCanceled,
                   PostOnly = true
               });
}
```

Updates

The `CoinbaseBrokerageModel` doesn't support limit order updates, but you can cancel an existing limit order and then create a new limit order with the desired arguments. For more information about this workaround, see the Workaround for Brokerages That Don't Support Updates .

Fills

The `CoinbaseBrokerageModel` uses the ImmediateFillModel .

Slippage

The `CoinbaseBrokerageModel` uses the NullSlippageModel .

Fees

The `CoinbaseBrokerageModel` uses the CoinbaseFeeModel .

Buying Power

The `CoinbaseBrokerageModel` uses the `CashBuyingPowerModel` . The brokerage model doesn't currently support margin trading.

Settlement

The `CoinbaseBrokerageModel` uses the ImmediateSettlementModel .

Margin Interest Rate

The `CoinbaseBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The default market of the `CoinbaseBrokerageModel` is `Market.Coinbase` .

Account Currency

The `CoinbaseBrokerageModel` doesn't set a default currency. To change the algorithm's currency from USD to a different currency, see Set Account Currency .

Supported Models

Bitfinex

Introduction

This page explains the `BitfinexBrokerageModel` , including the asset classes it supports, its default security-level models , and its default markets.

```C#
SetBrokerageModel(BrokerageName.Bitfinex, AccountType.Cash);
SetBrokerageModel(BrokerageName.Bitfinex, AccountType.Margin);
```

For more information about this model, see the class reference and implementation .

Asset Classes

The `BitfinexBrokerageModel` supports trading Crypto .

Orders

The `BitfinexBrokerageModel` supports several order types, order properties, and order updates.

Order Types

The following table describes the available order types for each asset class that the `BitfinexBrokerageModel` supports:

Order Type	Crypto
Market	✓
Limit	✓
Stop market	✓
Stop limit	✓

Order Properties

The `BitfinexBrokerageModel` supports custom order properties. The following table describes the members of the `BitfinexOrderProperties` object that you can set to customize order execution:

Property	Data Type	Description	Default Value
`TimeInForce`	`TimeInForce`	A TimeInForce instruction to apply to the order. The following instructions are supported: • `Day` • `GoodTilCanceled` • `GoodTilDate`	`TimeInForce.GoodTilCanceled`
`Hidden`	`bool`	A flag to signal that the order should be hidden. Hidden orders do not appear in the order book, so they do not influence other market participants. Hidden orders incur the taker fee.	
`PostOnly`	`bool`	A flag to signal that the order must only add liquidity to the order book and not take liquidity from the order book. If part of the order results in taking liquidity rather than providing liquidity, the order is rejected without any part of it being filled.	

C#

```
public override void Initialize()
{
    // Set the default order properties
    DefaultOrderProperties = new BitfinexOrderProperties
    {
        TimeInForce = TimeInForce.GoodTilCanceled,
        Hidden = false,
        PostOnly = false
    };
}

public override void OnData(Slice slice)
{
    // Use default order order properties
    LimitOrder(_symbol, quantity, limitPrice);

    // Override the default order properties
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new BitfinexOrderProperties
               {
                   TimeInForce = TimeInForce.Day,
                   Hidden = true,
                   PostOnly = false
               });
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new BitfinexOrderProperties
               {
                   TimeInForce = TimeInForce.GoodTilDate(new DateTime(year, month, day)),
                   Hidden = false,
                   PostOnly = true
               });
}
```

Updates

The `BitfinexBrokerageModel` supports order updates .

Fills

The `BitfinexBrokerageModel` uses the ImmediateFillModel .

Slippage

The `BitfinexBrokerageModel` uses the NullSlippageModel .

Fees

The `BitfinexBrokerageModel` uses the BitfinexFeeModel with the default argument values that models the current Bitfinex fee schedule.

Buying Power

The `BitfinexBrokerageModel` uses the `CashBuyingPowerModel` for cash accounts and the `SecurityMarginModel` for margin accounts.

If you have a margin account, the `BitfinexBrokerageModel` allows 3.3x leverage.

Settlement

The `BitfinexBrokerageModel` uses the ImmediateSettlementModel .

Margin Interest Rate

The `BitfinexBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The default market of the `BitfinexBrokerageModel` is `Market.Bitfinex` .

Account Currency

The `BitfinexBrokerageModel` doesn't set a default currency. To change the algorithm's currency from USD to a different currency, see Set Account Currency .

Supported Models

dYdX

Introduction

This page explains the `dYdXBrokerageModel` , including the asset classes it supports, its default security-level models , and its default markets.

```
SetBrokerageModel(BrokerageName.dYdX, AccountType.Margin);
```

For more information about this model, see the class reference and implementation .

Asset Classes

The `dYdXBrokerageModel` supports trading Crypto futures .

Orders

The `dYdXBrokerageModel` supports several order types, order properties, and order updates.

Order Types

The following table describes the available order types for each asset class that the `dYdXBrokerageModel` supports:

Order Type	Crypto Future
Market	✓
Limit	✓
Stop market	✓
Stop limit	✓

Order Properties

The `dYdXBrokerageModel` supports custom order properties. The following table describes the members of the `dYdXOrderProperties` object that you can set to customize order execution:

Property	Data Type	Description	Default Value
`TimeInForce`	`TimeInForce`	A TimeInForce instruction to apply to the order. The following instructions are supported: • `Day` • `GoodTilCanceled` • `GoodTilDate`	`TimeInForce.GoodTilCanceled`
`GasLimit`	`ulong`	The maximum amount of gas to use for the order.	1_000_000
`GoodTilBlockOffset`	`ulong`	The block height at which the order expires.	20
`PostOnly`	`bool`	A flag to signal that the order must only add liquidity to the order book and not take liquidity from the order book. If part of the order results in taking liquidity rather than providing liquidity, the order is rejected without any part of it being filled. This order property is only available for limit orders.	
`ReduceOnly`	`bool?`	A flag to signal that the order must only reduce your current position size. For more information about this order property, see Reduce-Only Order on the dYdX website.	

```C#
public override void Initialize()
{
    // Set the default order properties
    DefaultOrderProperties = new dYdXOrderProperties
    {
        TimeInForce = TimeInForce.GoodTilCanceled,
        ReduceOnly = false,
        PostOnly = false
    };
}

public override void OnData(Slice slice)
{
    // Use default order order properties
    LimitOrder(_symbol, quantity, limitPrice);

    // Override the default order properties
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new dYdXOrderProperties
               {
                   TimeInForce = TimeInForce.Day,
                   ReduceOnly = true,
                   PostOnly = false
               });
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new dYdXOrderProperties
               {
                   TimeInForce = TimeInForce.GoodTilDate(new DateTime(year, month, day)),
                   ReduceOnly = false,
                   PostOnly = true
               });
}
```

Updates

The `dYdXBrokerageModel` doesn't support order updates, but you can cancel an existing order and then create a new order with the desired arguments. For more information about this workaround, see the Workaround for Brokerages That Don't Support Updates .

Fills

The `dYdXBrokerageModel` uses the ImmediateFillModel .

Slippage

The `dYdXBrokerageModel` uses the NullSlippageModel .

Fees

The `dYdXBrokerageModel` uses the dYdXFeeModel with the default argument values that models the current dYdX fee schedule.

Buying Power

The `dYdXBrokerageModel` uses the `SecurityMarginModel` and allows 3.3x leverage.

Settlement

The `dYdXBrokerageModel` uses the ImmediateSettlementModel .

Margin Interest Rate

The `dYdXBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The default market of the `dYdXBrokerageModel` is `Market.dYdX` .

Account Currency

The `dYdXBrokerageModel` doesn't set a default currency. To change the algorithm's currency from USD to a different currency, see Set Account Currency .

Supported Models

Terminal Link

Introduction

This page explains the Terminal Link brokerage, including the asset classes it supports, its default security-level models , and its default markets.

Asset Classes

Terminal Link supports trading the following asset classes:

- Equities
- Equity Options
- Futures
- Index Options

Orders

Terminal Link enables you to create and manage Bloomberg™ orders.

Order Types

The following table describes the available order types for each asset class that Terminal Link supports:

Order Type	Equity	Equity Options	Futures	Index Options
Market	✓	✓	✓	✓
Limit	✓	✓	✓	✓
Stop market	✓	✓	✓	✓
Stop limit	✓	✓	✓	✓

Order Properties

We model custom order properties from the Bloomberg EMSX API. The following table describes the members of the `TerminalLinkOrderProperties` object that you can set to customize order execution:

Property	Data Type	Description	Default Value
`TimeInForce`	`TimeInForce`	A TimeInForce instruction to apply to the order. The following instructions are supported: • `Day` • `GoodTilCanceled` • `GoodTilDate`	`TimeInForce.GoodTilCanceled`

`Notes`	`string`	The free form instructions that may be sent to the broker.	
`HandlingInstruction`	`string`	The instructions for handling the order or route. The values can be preconfigured or a value customized by the broker.	
`CustomNotes1`	`string`	Custom user order notes 1. For more information about custom order notes, see Custom Notes & Free Text Fields in the EMSX API documentation	
`CustomNotes2`	`string`	Custom user order notes 2.	
`CustomNotes3`	`string`	Custom user order notes 3.	
`CustomNotes4`	`string`	Custom user order notes 4.	
`CustomNotes5`	`string`	Custom user order notes 5.	
`Account`	`string`	The EMSX account.	
`Broker`	`string`	The EMSX broker code.	
`Strategy`	`StrategyParameters`	An object that represents the EMSX order strategy details. You must append strategy parameters in the order that the EMSX API expects. The following strategy names are supported: "DMA", "DESK", "VWAP", "TWAP", "FLOAT", "HIDDEN", "VOLUMEINLINE", "CUSTOM", "TAP", "CUSTOM2", "WORKSTRIKE", "TAPNOW", "TIMED", "LIMITTICK", "STRIKE"	
`ExecutionInstruction`	`string`	The execution instruction field.	
`AutomaticPositionSides`	`bool`	A flag that determines whether to automatically include the position side in the order direction (buy-to-open, sell-to-close, etc.) instead of the default (buy, sell).	

PositionSide	OrderPosition?	An OrderPosition object that specifies the position side in the order direction (buy-to-open, sell-to-close, etc.) instead of the default (buy, sell). This member has precedence over AutomaticPositionSides .	
Exchange	Exchange	Defines the exchange name for sending the order to.	

```C#
public override void Initialize()
{
    // Set the default order properties
    DefaultOrderProperties = new TerminalLinkOrderProperties
    {
        TimeInForce = TimeInForce.GoodTilCanceled,
        Strategy = new TerminalLinkOrderProperties.StrategyParameters(
            "VWAP",
            new List<TerminalLinkOrderProperties.StrategyField>
            {
                new("09:30:00"),
                new("10:30:00"),
                new(),
                new()
            }
        )
    };
}

public override void OnData(Slice slice)
{
    // Use default order order properties
    LimitOrder(_symbol, quantity, limitPrice);

    // Override the default order properties
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new TerminalLinkOrderProperties
               {
                   TimeInForce = TimeInForce.Day,
                   Account = "account1"
               });
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new TerminalLinkOrderProperties
               {
                   TimeInForce = TimeInForce.GoodTilDate(new DateTime(year, month, day)),
                   Account = "account2"
               });
}
```

For more information about the format that the Bloomberg EMSX API expects, see Create Order and Route Extended Request in the EMSX API documentation and the createOrderAndRouteWithStrat documentation on the MathWorks website.

Get Open Orders

Terminal Link lets you access open orders .

Monitor Fills

Terminal Link allows you to monitor orders as they fill through order events .

Updates

Terminal Link doesn't support order updates , but you can cancel an existing order and then create a new order with the desired arguments. For more information about this workaround, see the Workaround for Brokerages That Don't Support Updates .

Cancellations

Terminal Link enables you to cancel open orders .

Handling Splits

If you're using raw data normalization and you have active orders with a limit, stop, or trigger price in the market for a US Equity when a stock split occurs, the following properties of your orders automatically adjust to reflect the stock split:

- Quantity
- Limit price
- Stop price
- Trigger price

Brokerage-Side Orders

By default, your algorithm doesn't record orders that you submit to your account by third-parties instead of through LEAN. To accept these orders, create a custom brokerage message handler .

Fills

To view how we model fills in backtests, see Fills .

Slippage

To view how we model slippage in backtests, see Slippage .

Fees

To view how we model fees in backtests, see Fees .

Buying Power

To view how we model buying power in backtests, see Buying Power .

Settlement

To view how we model settlements in backtests, see Settlement .

Margin Interest Rate

To view how we model margin interest rates in backtests, see Margin Interest Rate .

Default Markets

To view the default markets of each asset class when you use Terminal Link, see Default Markets .

Account Currency

The Terminal Link brokerage doesn't set a default currency. To change the algorithm's currency from USD to a different currency, see Set Account Currency .

Supported Models

SSC Eze

Introduction

This page explains the `EzeBrokerageModel` , including the asset classes it supports, its default security-level models , and its default markets.

```
SetBrokerageModel(BrokerageName.Eze, AccountType.Margin);
```

For more information about this model, see the class reference and implementation .

Asset Classes

The `EzeBrokerageModel` supports the following asset classes:

- US Equities
- Equity Options
- Futures
- Future Options
- Indices
- Index Options

Orders

The `EzeBrokerageModel` supports several order types, order properties, and order updates.

Order Types

The following table describes the available order types for each asset class that the `EzeBrokerageModel` supports:

Order Type	Equity	Equity Options	Futures	Future Options	Index Options
Market	✓	✓	✓	✓	✓
Limit	✓	✓	✓	✓	✓
Stop market	✓	✓	✓	✓	✓
Stop limit	✓	✓	✓	✓	✓
Market on Open	✓				
Market on Close	✓				

Order Properties

The `EzeBrokerageModel` supports custom order properties. The following table describes the members of the `EzeOrderProperties` object that you can set to customize order execution.

Property	Data Type	Description	Default Value
`TimeInForce`	`TimeInForce`	A TimeInForce instruction to apply to the order. The following instructions are supported: • `Day` • `GoodTilCanceled` • `GoodTilDate`	`TimeInForce.GoodTilCanceled`
`Account`	`string`	Sets a semi-colon separated list of trade or neutral accounts the user has permission for, e.g., "TAL;TEST;USER1;TRADE" or "TAL;TEST;USER2;NEUTRAL".	
`AccountType`	`string`	Sets the account type for the order. E.g., **"119"** for margin orders in Eze EMS.	
`Notes`	`string`	Sets the user message or notes.	
`Route`	`string`	Sets the route name as shown in SS&C Eze EMS.	

C#

```
public override void Initialize()
{
    // Set the default order properties
    DefaultOrderProperties = new EzeOrderProperties
    {
        TimeInForce = TimeInForce.GoodTilCanceled,
        Notes = "Default order properties"
    };
}

public override void OnData(Slice slice)
{
    // Use default order order properties
    LimitOrder(_symbol, quantity, limitPrice);

    // Override the default order properties
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new EzeOrderProperties
               {
                   TimeInForce = TimeInForce.Day,
                   Notes = "Default order properties"
               });
}
```

Updates

The `EzeBrokerageModel` supports order updates .

Handling Splits

If you're using raw data normalization and you have active orders with a limit, stop, or trigger price in the market for a US Equity when a stock split occurs, the following properties of your orders automatically adjust to reflect the stock split:

- Quantity

- Limit price
- Stop price
- Trigger price

Fills

The following table shows the fill model that the `EzeBrokerageModel` uses for each `SecurityType` :

`SecurityType`	**Fill Model**
`Equity`	EquityFillModel
`Future`	FutureFillModel
`FutureOption`	FutureOptionFillModel
Remaining `SecurityType` values	ImmediateFillModel

Slippage

The `EzeBrokerageModel` uses the NullSlippageModel .

Fees

The `EzeBrokerageModel` uses the EzeFeeModel which extends the ConstantFeeModel with with zero as `fee` for all assets.

Buying Power

The `EzeBrokerageModel` sets the buying power model based on the asset class of the security. The following table shows the default buying power model of each asset class:

Asset Class	**Model**
Equities	`SecurityMarginModel`
Futures	`FutureMarginModel`
All Options Types	`OptionMarginModel`

If you have a margin account, the `EzeBrokerageModel` allows 2x leverage for Equities. The remaining asset classes are derivatives.

Settlement

The following table shows which settlement model the `EzeBrokerageModel` uses based on the security type and your account type:

Security Type	Account Type	Settlement Model
Equity	Cash	DelayedSettlementModel with the default settlement rules
Option	Cash	DelayedSettlementModel with the default settlement rules
Future	Any	FutureSettlementModel

For all other cases, the `EzeBrokerageModel` uses the ImmediateSettlementModel .

```C#
// For US Equities with a cash account:
security.SetSettlementModel(new DelayedSettlementModel(Equity.DefaultSettlementDays, Equity.DefaultSettlementTime));

// For Equity Options with a cash account:
security.SetSettlementModel(new DelayedSettlementModel(Option.DefaultSettlementDays, Option.DefaultSettlementTime));

// For Futures
security.SetSettlementModel(new FutureSettlementModel());

// For remaining cases:
security.SetSettlementModel(new ImmediateSettlementModel());
```

Margin Interest Rate

The `EzeBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The following table describes the default markets of each `SecurityType` for the `EzeBrokerageModel` :

SecurityType	Market
Equity	USA
Option	USA
Future	CME
FutureOption	CME
Index	USA
IndexOption	USA

Supported Models

Trading Technologies

Introduction

This page explains the `TradingTechnologiesBrokerageModel`, including the asset classes it supports, its default security-level models, and its default markets.

```C#
SetBrokerageModel(BrokerageName.TradingTechnologies, AccountType.Cash);
SetBrokerageModel(BrokerageName.TradingTechnologies, AccountType.Margin);
```

For more information about this model, see the class reference and implementation.

Asset Classes

The `TradingTechnologiesBrokerageModel` supports trading Futures.

Orders

The `TradingTechnologiesBrokerageModel` supports several order types, some `TimeInForce` order properties, and order updates.

Order Types

The following table describes the available order types for each asset class that the `TradingTechnologiesBrokerageModel` supports:

Order Type	Futures
Market	✓
Limit	✓
Stop market	✓
Stop limit	✓

The `TradingTechnologiesBrokerageModel` enforces the following order rules:

- If you are buying (selling) with a `StopMarketOrder` or a `StopLimitOrder`, the stop price of the order must be greater (less) than the current security price.
- If you are buying (selling) with a `StopLimitOrder`, the limit price of the order must be greater (less) than the stop price.

Time In Force

The `TradingTechnologiesBrokerageModel` supports the `Day` and `GoodTilCanceled` TimeInForce order properties.

```
C#
public override void Initialize()
{
    // Set the default order properties
    DefaultOrderProperties.TimeInForce = TimeInForce.GoodTilCanceled;
}

public override void OnData(Slice slice)
{
    // Use default order order properties
    LimitOrder(_symbol, quantity, limitPrice);

    // Override the default order properties
    LimitOrder(_symbol, quantity, limitPrice,
               orderProperties: new OrderProperties
               {
                   TimeInForce = TimeInForce.Day
               });
}
```

Updates

The `TradingTechnologiesBrokerageModel` supports order updates .

Fills

The `TradingTechnologiesBrokerageModel` uses the FutureFillModel .

Slippage

The `TradingTechnologiesBrokerageModel` uses the NullSlippageModel .

Fees

The `TradingTechnologiesBrokerageModel` uses the ConstantFeeModel with zero fees.

```
C#
security.SetFeeModel(new ConstantFeeModel(0.0m));
```

Buying Power

The `TradingTechnologiesBrokerageModel` uses the `FutureMarginModel` .

Settlement

The `TradingTechnologiesBrokerageModel` uses the FutureSettlementModel .

Margin Interest Rate

The `TradingTechnologiesBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The default market of the `TradierBrokerageModel` is `Market.CME` .

Account Currency

The `TradingTechnologiesBrokerageModel` doesn't set a default currency. To change the algorithm's currency from USD to a different currency, see Set Account Currency .

Supported Models

Wolverine

Introduction

This page explains the `WolverineBrokerageModel` , including the asset classes it supports, its default security-level models , and its default markets.

```
C#

SetBrokerageModel(BrokerageName.Wolverine, AccountType.Cash);
SetBrokerageModel(BrokerageName.Wolverine, AccountType.Margin);
```

For more information about this model, see the class reference and implementation .

Asset Classes

The `WolverineBrokerageModel` supports trading US Equities .

Orders

The `WolverineBrokerageModel` supports one order type, but doesn't support order updates or extended market hours trading.

Order Types

The `WolverineBrokerageModel` supports market orders .

Updates

The `WolverineBrokerageModel` doesn't support order updates.

Extended Market Hours

The `WolverineBrokerageModel` doesn't support extended market hours trading. If you place an order outside of regular trading hours, the order is invalid.

Order Properties

The `WolverineBrokerageModel` supports custom order properties. The following table describes the members of the `WolverineOrderProperties` object that you can set to customize order execution:

Property	Description

`TimeInForce`	`TimeInForce`

Exchange	Exchange

`ExchangePostFix`	`string`

```C#
public override void Initialize()
{
    // Set the default order properties
    DefaultOrderProperties = new WolverineOrderProperties
    {
        TimeInForce = TimeInForce.GoodTilCanceled,
        Exchange = Exchange.SMART,
        ExchangePostFix = "-INCA-TX"
    };
}
```

Fills

The `WolverineBrokerageModel` uses the EquityFillModel for Equity trades.

Slippage

The `WolverineBrokerageModel` uses the NullSlippageModel .

Fees

The `WolverineBrokerageModel` uses the WolverineFeeModel .

Buying Power

The `WolverineBrokerageModel` uses the `SecurityMarginModel` . If you have a margin account, the `WolverineBrokerageModel` allows up to 2x leverage.

Settlement

The `WolverineBrokerageModel` uses the ImmediateSettlementModel for margin accounts and the DelayedSettlementModel with the default settlement rules for cash accounts.

```
C#

// For cash accounts:
security.SetSettlementModel(new DelayedSettlementModel(Equity.DefaultSettlementDays, Equity.DefaultSettlementTime));

// For margin accounts:
security.SetSettlementModel(new ImmediateSettlementModel());
```

Margin Interest Rate

The `WolverineBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The default market of the `WolverineBrokerageModel` is `Market.USA` .

Account Currency

The `WolverineBrokerageModel` doesn't set a default currency. To change the algorithm's currency from USD to a different currency, see Set Account Currency .

Supported Models

Oanda

Introduction

This page explains the `OandaBrokerageModel` , including the asset classes it supports, its default security-level models , and its default markets.

```
SetBrokerageModel(BrokerageName.OandaBrokerage, AccountType.Margin);
```

For more information about this model, see the class reference and implementation .

Asset Classes

The `OANDABrokerageModel` supports trading Forex and CFDs .

Orders

The `OandaBrokerageModel` supports several order types, the `TimeInForce` order property, and order updates.

Order Types

The following table describes the available order types for each asset class that the `OANDABrokerageModel` supports:

Order Type	Forex	CFD
Market	✓	✓
Limit	✓	✓
Stop market	✓	✓
Stop limit	✓	✓

Time In Force

The `OANDABrokerageModel` the `GoodTilCanceled` TimeInForce .

```
DefaultOrderProperties.TimeInForce = TimeInForce.GoodTilCanceled;
LimitOrder(_symbol, quantity, limitPrice);
```

Updates

The `OANDABrokerageModel` supports order updates .

Fills

The `OandaBrokerageModel` uses the ImmediateFillModel .

Slippage

The `OandaBrokerageModel` uses the NullSlippageModel .

Fees

The `OandaBrokerageModel` uses the ConstantFeeModel with zero fees. OANDA offers "Spread" and "Core + Commission" pricing models. Our data is the spread dataset, so we model the spread pricing. To estimate OANDA Core+Commission data, use a custom fee model and a custom fill model .

```
C#
security.SetFeeModel(new ConstantFeeModel(0.0m));
```

Buying Power

The `OandaBrokerageModel` uses the `SecurityMarginModel` and allows up to 50x leverage.

Settlement

The `OandaBrokerageModel` uses the ImmediateSettlementModel for Forex trades and the AccountCurrencyImmediateSettlementModel for CFD trades.

Margin Interest Rate

The `OandaBrokerageModel` uses the NullMarginInterestRateModel .

Default Markets

The default market of the `OandaBrokerageModel` is `Market.Oanda` .

Account Currency

The `OandaBrokerageModel` doesn't set a default currency. To change the algorithm's currency from USD to a different currency, see Set Account Currency .

Reality Modeling

Brokerage Message Handler

Introduction

When your brokerage sends a message, you receive this information in the algorithm and can take any actions. When you place an order through your brokerage platform instead of placing it through LEAN, you can decide whether the transaction handler should process the order. By default, the transaction handler only process orders that you place through LEAN. Custom brokerage message handlers enable you to instruct the transaction handler to process orders that you create directly through your brokerage's platform.

Set Models

To set a brokerage message handler, in the `Initialize` method, call the `SetBrokerageMessageHandler` method.

```C#
public override void Initialize()
{
    // Set a custom brokerage message handler for the algorithm in Initialize method
    // It handles different type of returned message from the broker, helping you filter or abstract for the ones you
care about
    SetBrokerageMessageHandler(new MyBrokerageMessageHandler(this));
}
```

Default Behavior

The default brokerage message handler is the `DefaultBrokerageMessageHandler` . The following table describes how the `DefaultBrokerageMessageHandler` processes brokerage messages:

Brokerage Message Type	Action
`BrokerageMessageType. Information`	Sends a debug message for the algorithm.
`BrokerageMessageType. Warning`	Sends an error message for the algorithm.
`BrokerageMessageType. Error`	Sets the algorithm runtime error and stops it.
`BrokerageMessageType. Disconnect`	Stops the algorithm after 15 minutes if the market is open. Otherwise, it stops five minutes after market open.
`BrokerageMessageType. Reconnect`	Cancels the disconnection process.

The default brokerage message handler rejects orders you create through the brokerage. An example of this is when you place an order using the brokerage website or a third-party software.

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`algorithm`	`IAlgorithm`	The running algorithm.	
`initialDelay`	`TimeSpan?`	The amount of time LEAN will wait after a brokerage disconnection message for a reconnection message. If the reconnection message doesn't arrive before the time limit, LEAN stops running. If you don't provide a value, it uses `TimeSpan.FromMinutes(15)`.	`null`
`openThreshold`	`TimeSpan?`	Defines how long before market open to re-check for brokerage reconnect message. If you don't provide a value, it uses `TimeSpan.FromMinutes(5)`.	`null`

For more information about this model, see the class reference and implementation .

Model Structure

Brokerage message handlers extend the `DefaultBrokerageMessageHandler` class. Extensions of the `DefaultBrokerageMessageHandler` class should define the `HandleMessage` and `HandleOrder` methods.

The `HandleMessage` method processes the brokerage message event. It triggers any actions in the algorithm or notifications system required.

The `HandleOrder` method defines whether the transaction handler should process a new order you placed directly through the brokerage's website or third-party software.

```C#
public class CustomBrokerageMessageHandlerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the brokerage message handler
        SetBrokerageMessageHandler(new MyBrokerageMessageHandler(this));
    }
}

// Define the custom brokerage message handler
public class MyBrokerageMessageHandler : DefaultBrokerageMessageHandler
{
    private readonly IAlgorithm _algorithm;
    public MyBrokerageMessageHandler(IAlgorithm algorithm)
        : base(algorithm)
    {
        _algorithm = algorithm;
    }

    public void HandleMessage(BrokerageMessageEvent message)
    {
        _algorithm.Debug($"{_algorithm.Time.ToStringInvariant("o")} Event: {message.Message}");
    }

    public bool HandleOrder(NewBrokerageOrderNotificationEventArgs eventArgs)
    {
        return false;
    }
}
```

The `BrokerageMessageEvent` class has the following properties:

The `NewBrokerageOrderNotificationEventArgs` class has the following properties:

Brokerage Support

Most QuantConnect brokerage integrations don't support brokerage-side orders. If you create an order directly through your brokerage instead of through LEAN, your algorithm won't process the order because the return value of the `HandleOrder` method is ignored.

Terminal Link is the exception that can support brokerage-side orders. This algorithm demonstrates how to implement a custom brokerage handler that accepts orders with a target `TerminalLinkOrderProperties.CustomNotes1` value.

Examples

The following examples demonstrate common practices for implementing a custom brokerage message handler.

Example 1: Notification

The following algorithm buys and holds SPY. It implemented a custom brokerage message handler to send notifications to users through email and SMS in cases of disconnection, reconnections, and outside trades.

C#

```
public class CustomBrokerageMessageHandlerAlgorithm : QCAlgorithm
{
    private Symbol _spy;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // To simulate the connection to IB brokerage.
        SetBrokerageModel(BrokerageName.InteractiveBrokersBrokerage, AccountType.Margin);
        // Set the brokerage message handler to our custom model.
        SetBrokerageMessageHandler(new CustomBrokerageMessageHandler(this));

        // Request SPY data for trading.
        _spy = AddEquity("SPY").Symbol;
    }

    public override void OnData(Slice slice)
    {
        // Buy and hold SPY as a strategy.
        if (!Portfolio.Invested)
        {
            SetHoldings(_spy, 1m);
        }
    }
}

// Define the custom brokerage message handler.
public class CustomBrokerageMessageHandler : DefaultBrokerageMessageHandler
{
    private readonly IAlgorithm _algorithm;
    public CustomBrokerageMessageHandler(IAlgorithm algorithm)
        : base(algorithm)
    {
        _algorithm = algorithm;
    }

    public void HandleMessage(BrokerageMessageEvent message)
    {
        // Notify the user by SMS on disconnection of the algorithm instance from the brokerage.
        if (message.Type == BrokerageMessageType.Disconnect)
        {
            _algorithm.Notify.Sms(
                phoneNumber: "<YOUR_PHONE_NUMBER>",
                message: "Your QC Algorithm instance 'CustomBrokerageMessageHandlerAlgorithm' was disconnected from
the IB brokerage."
            );
        }
        // Notify the user by SMS on reconnection of the algorithm instance to the brokerage after disconnection.
        else if (message.Type == BrokerageMessageType.Reconnect)
        {
            _algorithm.Notify.Sms(
                phoneNumber: "<YOUR_PHONE_NUMBER>",
                message: "Your QC Algorithm instance 'CustomBrokerageMessageHandlerAlgorithm' was reconnected to the
IB brokerage."
            );
        }
    }

    public bool HandleOrder(NewBrokerageOrderNotificationEventArgs eventArgs)
    {
        // Notify the user by email on receiving an order from outside the algorithm.
        _algorithm.Notify.Email(
            address: "<YOUR_EMAIL_ADDRESS@example.com>",
            subject: "Outside Order",
            message: "Your QC Algorithm instance 'CustomBrokerageMessageHandlerAlgorithm' detected an outside order
from the IB brokerage; it has been processed to the algorithm."
        );
        // Process a new order placed through the brokerage's website or third-party software to handle it in the
algorithm instance.
        return true;
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
CustomBrokerageSideOrderHandlingRegressionAlgorithm.cs C# CustomBrokerageMessageHandlerAlgorithm.cs C#

Reality Modeling

Buying Power

Introduction

Buying power models (also known as margin models) control how much buying power or leverage your portfolio has to make trades. When you place an order, LEAN uses the buying power model to determine whether the order should be submitted so you avoid placing orders that would be rejected by the brokerage. Buying power calculations can be very complex and depend on many factors, including the brokerage or even the time of day. For example, the `PatternDayTradingMarginModel` lets you have 4x leverage during regular trading hours and 2x leverage during pre-market and post-market trading hours.

What is Buying Power?

Buying power is the total amount of money you can spend in your brokerage account. It depends on the security type and account type. On one hand, Option and Future contracts are leveraged securities with specific buying power rules. On the other hand, the buying power of cash accounts is just the cash in the account while the buying power of margin accounts is leveraged by the brokerage credit.

What is Margin?

Margin is a credit loan from your brokerage you receive after you deposit collateral into your margin account. You need margin to place short-biased trades and you can use margin to increase your buying power, but you incur interest fees. Margin is the dollar value of the loan that the brokerage gives you. A margin requirement of 25% means that to purchase 10,000 worth of securities, you need at least 2,500 worth of collateral in your brokerage account to open the trade and you can borrow the rest on margin. Maintenance margin is the minimum equity (equity = total portfolio value - borrowed funds) you must have in your brokerage account to stay in your positions. If the value of your portfolio falls below the maintenance margin, you receive a margin call . If you receive a margin call, you either need to add more capital to your brokerage account or the brokerage will liquidate some of your holdings to reduce your exposure and their risk.

Some securities have special margin rules. Derivatives are leveraged assets with a floating margin requirement. In particular, long Options have zero maintenance margin requirement and their initial margin requirement is only the premium that you pay upfront.

What is Leverage?

Leverage is using borrowed money to increase your buying power. Leverage has an inverse relationship with your margin requirement and maintenance margin. If you have a margin requirement of 50%, you can use up to 1 / 50% = 2 leverage. Trading with leverage can be risky. It can boost your returns on profitable trades but can make your losing trades more expensive. If you have 10,000 in your brokerage margin account and purchase 20,000 worth of securities, you are trading with 2x leverage. If the value of the securities in your portfolio drops by 50% when you have a 2x leverage position, you lose all of your equity.

What Are Position Groups?

A position group is the combination of holdings. It has lower margin requirement and maintenance margin than the sum of each position of the group. If you have a margin requirement of 29,150 to purchase an in-the-money call Option contract, and a margin requirement of 101,499 to sell an out-of-the money call Option contract, the total margin requirement is $130,649. However, these positions compose a bull call spread with a margin requirement of $0.

Set Models

The brokerage model of your algorithm automatically sets the buying power model for each security, but you can override it. To manually set the buying power model of a security, call the `SetBuyingPowerModel` method on the `Security` object.

```
C#
public override void Initialize()
{
    var security = AddEquity("SPY");
    // Use 3x leverage on the selected security, which mimic the actual buying power of your broker
    security.SetBuyingPowerModel(new SecurityMarginModel(3m));
}
```

You can also set the buying power model in a security initializer . If your algorithm has a universe, use the security initializer technique. In order to initialize single security subscriptions with the security initializer, call `AddSecurityInitializer` before you create the subscriptions.

```
C#
public class AddSecurityInitializerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the security initializer to set models of assets.
        AddSecurityInitializer(CustomSecurityInitializer);
    }

    private void CustomSecurityInitializer(Security security)
    {
        // Overwrite some of the reality models
        security.SetBuyingPowerModel(new SecurityMarginModel(3m));
    }
}
```

You cannot change the position group buying power models.

Supported Models

The following buying power models are available:

- BuyingPowerModel
- CashBuyingPowerModel
- ConstantBuyingPowerModel
- CryptoFutureMarginModel
- NullBuyingPowerModel
- FutureMarginModel
- FuturesOptionsMarginModel
- SecurityMarginModel
- OptionMarginModel
- PatternDayTradingMarginModel

The following position group buying power models are available:

- SecurityPositionGroupBuyingPowerModel
- OptionStrategyPositionGroupBuyingPowerModel

Default Behavior

The brokerage model of your algorithm automatically sets the buying power model of each security. The default brokerage

model is the `DefaultBrokerageModel`, which sets the buying power model based on the asset class of the security. The following table shows the default buying power model of each asset class:

Asset Class	Model
Equity Options	`OptionMarginModel`
Futures	`FutureMarginModel`
Future Options	`FuturesOptionsMarginModel`
Index Options	`OptionMarginModel`
Crypto	`CashBuyingPowerModel` for cash accounts or `SecurityMarginModel` for margin accounts
CryptoFuture	`CryptoFutureMarginModel`
Forex	`CashBuyingPowerModel` for cash accounts or `SecurityMarginModel` for margin accounts
Other	`SecurityMarginModel`

Model Structure

Buying power models should extend the `BuyingPowerModel` class. Extensions of the `BuyingPowerModel` class should implement the following methods:

```C#
public class CustomBuyingPowerModelExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        var security = AddEquity("SPY");
        // Set the custom buying power model of the selected security to mimic its actual buying power in your used
broker
        security.SetBuyingPowerModel(new MyBuyingPowerModel());
    }
}

// Define the custom buying power model
class MyBuyingPowerModel : BuyingPowerModel
{
    public MyBuyingPowerModel(
        decimal leverage = 2m,
        decimal requiredFreeBuyingPowerPercent = 0m)
        : base(leverage, requiredFreeBuyingPowerPercent)
    {
    }

    public override decimal GetLeverage(Security security)
    {
       return base.GetLeverage(security);
    }

    public override void SetLeverage(Security security, decimal leverage)
    {
        base.SetLeverage(security, leverage);
    }

    public override InitialMargin GetInitialMarginRequiredForOrder(
        InitialMarginRequiredForOrderParameters parameters)
    {
        return base.GetInitialMarginRequiredForOrder(parameters);
    }

    public override MaintenanceMargin GetMaintenanceMargin(
        MaintenanceMarginParameters parameters)
```

```
        MaintenanceMarginParameters parameters)
    {
        return base.GetMaintenanceMargin(parameters);
    }

    protected override decimal GetMarginRemaining(
        SecurityPortfolioManager portfolio,
        Security security,
        OrderDirection direction)
    {
        return base.GetMarginRemaining(portfolio, security, direction);
    }

    public override InitialMargin GetInitialMarginRequirement(
        InitialMarginParameters parameters)
    {
        return base.GetInitialMarginRequirement(parameters);
    }

    public override HasSufficientBuyingPowerForOrderResult
HasSufficientBuyingPowerForOrder(HasSufficientBuyingPowerForOrderParameters parameters)
    {
        return base.HasSufficientBuyingPowerForOrder(parameters);
    }

    public override GetMaximumOrderQuantityResult
GetMaximumOrderQuantityForDeltaBuyingPower(GetMaximumOrderQuantityForDeltaBuyingPowerParameters parameters)
    {
        return base.GetMaximumOrderQuantityForDeltaBuyingPower(parameters);
    }

    public override GetMaximumOrderQuantityResult
GetMaximumOrderQuantityForTargetBuyingPower(GetMaximumOrderQuantityForTargetBuyingPowerParameters parameters)
    {
        return base.GetMaximumOrderQuantityForTargetBuyingPower(parameters);
    }

    public override ReservedBuyingPowerForPosition GetReservedBuyingPowerForPosition(
        ReservedBuyingPowerForPositionParameters parameters)
    {
        return base.GetReservedBuyingPowerForPosition(parameters);
    }

    public override BuyingPower GetBuyingPower(BuyingPowerParameters parameters)
    {
        return base.GetBuyingPower(parameters);
    }
}
```

Disable Buying Power Models

You can disable order margin checks and opt to let your brokerage decide to accept or reject the trades. This is helpful in live trading if you have a more permissive brokerage margin allowance that what LEAN models. The default position group buying power models are helpful for Option trading strategies. However, it can be counterproductive if it's not a supported Option strategy . To disable the validations of the default position group buying power model, use the `NullSecurityPositionGroupModel` . To set the `NullSecurityPositionGroupModel` for the portfolio, during initialization , call the `SetPositions` method with the `SecurityPositionGroupModel. Null` argument.

```
C#
Portfolio.SetPositions(SecurityPositionGroupModel.Null);
```

To disable the validations of the default buying power model , use the `NullBuyingPowerModel` . To set the `NullBuyingPowerModel` for a security subscription, call the `SetBuyingPowerModel` method with the `BuyingPowerModel. Null` argument.

```
C#
public override void Initialize()
{
    var equity = AddEquity("SPY");
    // To disable any buying power of the selected security
    equity.SetBuyingPowerModel(BuyingPowerModel.Null);
    // Alias:
    // equity.SetMarginModel(SecurityMarginModel.Null);
}
```

You can also set the `NullBuyingPowerModel` in a security initializer . If your algorithm has a universe, use the security initializer technique. To set the buying power of securities in the security initializer, set the brokerage model , set the security initializer , and then create security subscriptions.

```
C#
public class AddSecurityInitializerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the security initializer to set models of assets.
        AddSecurityInitializer(CustomSecurityInitializer);
    }

    private void CustomSecurityInitializer(Security security)
    {
        // Overwrite the security buying power
        security.SetBuyingPowerModel(BuyingPowerModel.Null);
    }
}
```

Set Asset Leverage

The buying power model sets the leverage for each security in your algorithm, but you can override its leverage settings after the buying power model is set.

To set the leverage when you create a security subscription, pass in a `leverage` argument.

```
C#
public override void Initialize()
{
 // Set the leverage to 3x manually subjected to specific need
 AddEquity("SPY", leverage: 3);
}
```

You can also set the asset leverage in a security initializer. In order to set the leverage of securities in the security initializer, call `AddSecurityInitializer` before you create security subscriptions and after you call `SetBrokerageModel` . If you pass in a `leverage` argument when you create the security subscription, the `leverage` argument takes precedence over the `SetLeverage` call in the security initializer.

```C#
public class AddSecurityInitializerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the security initializer to set models of assets.
        AddSecurityInitializer(CustomSecurityInitializer);
    }

    private void CustomSecurityInitializer(Security security)
    {
        // Overwrite the security leverage
        security.SetLeverage(3);
    }
}
```

To set the leverage for all securities in a universe, set the `UniverseSettings.Leverage` property.

```C#
public override void Initialize()
{
 // Set the leverage to 3x to all securities, it should only be used for a very narrow spectrum of security universe
with the same leverage
 // E.g. US Equities universe composed of only primary stocks in NYSE exchange
 UniverseSettings.Leverage = 3;
}
```

In live trading, LEAN doesn't ignore the leverage you set. However, if you set a different leverage from what your brokerage provides, it creates a mismatch between the buying power in your algorithm and the buying power the brokerage gives you. In this case, orders can pass the validations in LEAN but your brokerage may reject them.

PDT Rule

If all of the following statements are true, you are classified as a pattern day trader:

- You reside in the United States.
- You trade in a margin account.
- You execute 4+ intraday US Equity trades within 5 business days.
- Your intraday US Equity trades represent more than 6% of your total trades.

Pattern day traders must maintain a minimum equity of $25,000 in their margin account to continue trading. For more information about pattern day trading, see Am I a Pattern Day Trader? on the FINRA website.

The `PatternDayTradingMarginModel` doesn't enforce minimum equity rules and doesn't limit your trades, but it adjusts your available leverage based on the market state. During regular market hours, you can use up to 4x leverage. During extended market hours, you can use up to 2x leverage.

```C#
security.MarginModel = new PatternDayTradingMarginModel();
```

Get Initial Margin Requirements

The following sections explain how to check if you have enough margin remaining to cover the initial margin requirements of various order types.

Check Requirements of Regular Orders

Follow these steps to check if you have enough margin remaining to cover the initial margin requirements of an order:

1. Create an `InitialMarginParameters` object with the security and quantity you want to trade.

```
C#

public override void OnData(Slice slice)
{
    var security = Securities["SPY"];
    var quantity = 100m;
    var parameter = new InitialMarginParameters(security, quantity);
```

2. Call the `GetInitialMarginRequirement` method of the security's buying power model with the `InitialMarginParameters` .

```
C#

    var initialMargin = security.BuyingPowerModel.GetInitialMarginRequirement(parameter);
```

   The `GetInitialMarginRequirement` method returns an `InitialMargin` object, which have the following properties:

3. Compare the margin you have remaining against the initial margin requirement of the order.

```
C#

    if (Portfolio.MarginRemaining >= initialMargin.Value)
    {
        MarketOrder(security.Symbol, quantity);
    }
    else
    {
        Debug("You don't have sufficient margin for this order.");
    }
}
```

Check Requirements of Option Strategy Orders

Follow these steps to check if you have enough margin remaining to cover the initial margin requirements of an Option strategy :

1. Create an `OptionStrategy` object with the strategy you want to trade and its buying power model.

   For example, create a Bull Put Spread strategy.

```
C#

private Symbol _symbol;

public override void Initialize()
{
    // Subscribe to option data and cache the canonical symbol to obtain the option data
    _symbol = AddOption("SPY").Symbol;
}

public override void OnData(Slice slice)
{
    // Trade on updated option chain data
    if (!slice.OptionChains.TryGetValue(_symbol, out var chain))
    {
        return;
    }

    var itmStrike = chain.Max(x => x.Strike);
    var otmStrike = chain.Min(x => x.Strike);
    var expiry = chain.Min(x => x.Expiry);

    var optionStrategy = OptionStrategies.BullPutSpread(_symbol, itmStrike, otmStrike, expiry);
```

2. Create an `OptionStrategyPositionGroupBuyingPowerModel` object of the strategy.

```
C#
var buyingPowerModel = new OptionStrategyPositionGroupBuyingPowerModel(optionStrategy);
```

3. Create a list of `Position` objects from the strategy legs.

```
C#
var positions = optionStrategy.OptionLegs.Select(leg =>
{
    var symbol = QuantConnect.Symbol.CreateOption(_symbol.Underlying, _symbol.ID.Market,
_symbol.ID.OptionStyle, leg.Right, leg.Strike, leg.Expiration);
    return new Position(symbol, leg.Quantity, 1);
});
```

4. Create a `PositionGroupInitialMarginParameters` objects with the portfolio and the position group.

```
C#
var positionGroup = new PositionGroup(buyingPowerModel, 1, positions.ToArray());
var parameters = new PositionGroupInitialMarginParameters(Portfolio, positionGroup);
```

5. Call the `GetInitialMarginRequirement` method of the strategy's buying power model with the `PositionGroupInitialMarginParameters`.

```
C#
var initialMargin = buyingPowerModel.GetInitialMarginRequirement(parameters);
```

   The `GetInitialMarginRequirement` method returns an `InitialMargin` object, which have the following properties:

6. Compare the margin you have remaining against the initial margin requirement of the order.

   In this case, the `initialMargin.Value` is the initial margin requirement for a trade size of 1. For different trade sizes, multiply the `initialMargin.Value` by the quantity.

```
C#
var strategyQuantity = 2;
if (Portfolio.MarginRemaining >= strategyQuantity * initialMargin.Value)
{
    Buy(optionStrategy, quantity);
}
else
{
    Debug("You don't have sufficient margin for this order.");
}
}
```

Examples

The following examples demonstrate some common practices for implementing a custom buying power model.

Example 1: Always Sufficient Buying Power

The following algorithm trades protective call strategy. Sometimes, the brokerage's margin requirement differs from the algorithm's default. To avoid errors on order submission at the algorithm level, we can use the `SecurityPositionGroupModel.Null` and `NullBuyingPowerModel` to disable buying power limitation so the order is sent directly to the broker.

```C#
using QuantConnect.Securities.Positions;

public class BuyingPowerModelAlgorithm : QCAlgorithm
{
    private Symbol _spy;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        Settings.SeedInitialPrices = true;
        AddSecurityInitializer(security =ggt;
        {
            // Do not allow buying power for options; only a hedging strategy with 0 margin is allowed.
            security.SetBuyingPowerModel(new NullBuyingPowerModel());
        });

        // Disable the validations of the default position group buying power model
        // that evaluates the buying power of multi-leg strategies
        Portfolio.SetPositions(SecurityPositionGroupModel.Null);

        // Request SPY data for trading. Set the data normalization mode to raw to allow for a fair comparison in
strike price.
        _spy = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;

        // Set a scheduled event to rebalance the protective put strategy.
        Schedule.On(
            DateRules.WeekStart(_spy),
            TimeRules.AfterMarketOpen(_spy, 0),
            Rebalance
        );
    }

    private void Rebalance()
    {
        // Select the weekend expiring ATM put for the protective put strategy.
        var atmPutContract = OptionChain(_spy)
            .Where(x => x.Right == OptionRight.Put && x.Expiry < Time.AddDays(6))
            .OrderByDescending(x => x.Expiry)
            .ThenBy(x => Math.Abs(x.Strike - x.UnderlyingLastPrice))
            .First();
        // Request the ATM put data for trading.
        var atmPut = AddOptionContract(atmPutContract).Symbol;

        if (Portfolio[_spy].Invested)
        {
            // If SPY is not assigned, we only need to buy the put.
            MarketOrder(atmPut, 1, tag: "Protective Put");
            return;
        }

        // Order the protective put strategy by combo order.
        ComboMarketOrder(
            new List<Leg> () {
                Leg.Create(_spy, 100),
                Leg.Create(atmPut, 1)
            },
            1,
            tag: "Protective Put"
        );
    }
}
```

Example 2: Default Model Wrapper

The following algorithm defines a subclass of the default `BuyingPowerModel` and overrides its built-in members so you can easily customize the parts you need.

```C#
public class CustomBuyingPowerModelAlgorithm : QCAlgorithm
{

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
```

```
        SetStartDate(2024, 7, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var security = AddEquity("SPY", Resolution.Daily);
        security.SetBuyingPowerModel(new MyBuyingPowerModel());
    }

    public override void OnData(Slice data)
    {
        if (!Portfolio.Invested)
        {
            SetHoldings("SPY", 1);
        }
    }
}

class MyBuyingPowerModel : BuyingPowerModel
{
    public MyBuyingPowerModel(
        decimal leverage = 2m,
        decimal requiredFreeBuyingPowerPercent = 0m)
        : base(leverage, requiredFreeBuyingPowerPercent)
    {
    }

    public override decimal GetLeverage(Security security)
    {
    return base.GetLeverage(security);
    }

    public override void SetLeverage(Security security, decimal leverage)
    {
        base.SetLeverage(security, leverage);
    }

    public override InitialMargin GetInitialMarginRequiredForOrder(
        InitialMarginRequiredForOrderParameters parameters)
    {
        return base.GetInitialMarginRequiredForOrder(parameters);
    }

    public override MaintenanceMargin GetMaintenanceMargin(
        MaintenanceMarginParameters parameters)
    {
        return base.GetMaintenanceMargin(parameters);
    }

    protected override decimal GetMarginRemaining(
        SecurityPortfolioManager portfolio,
        Security security,
        OrderDirection direction)
    {
        return base.GetMarginRemaining(portfolio, security, direction);
    }

    public override InitialMargin GetInitialMarginRequirement(
        InitialMarginParameters parameters)
    {
        return base.GetInitialMarginRequirement(parameters);
    }

    public override HasSufficientBuyingPowerForOrderResult
HasSufficientBuyingPowerForOrder(HasSufficientBuyingPowerForOrderParameters parameters)
    {
        return base.HasSufficientBuyingPowerForOrder(parameters);
    }

    public override GetMaximumOrderQuantityResult
GetMaximumOrderQuantityForDeltaBuyingPower(GetMaximumOrderQuantityForDeltaBuyingPowerParameters parameters)
    {
        return base.GetMaximumOrderQuantityForDeltaBuyingPower(parameters);
    }

    public override GetMaximumOrderQuantityResult
GetMaximumOrderQuantityForTargetBuyingPower(GetMaximumOrderQuantityForTargetBuyingPowerParameters parameters)
    {
        return base.GetMaximumOrderQuantityForTargetBuyingPower(parameters);
    }

    public override ReservedBuyingPowerForPosition GetReservedBuyingPowerForPosition(
        ReservedBuyingPowerForPositionParameters parameters)
```

```
        ReservedBuyingPowerForPositionParameters parameters)
    {
        return base.GetReservedBuyingPowerForPosition(parameters);
    }

    public override BuyingPower GetBuyingPower(BuyingPowerParameters parameters)
    {
        return base.GetBuyingPower(parameters);
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
CustomBuyingPowerModelAlgorithm.cs C# CustomModelsAlgorithm.cs C# NullBuyingPowerOptionBullCallSpreadAlgorithm.cs C#

Reality Modeling

Settlement

Settlement

Key Concepts

Introduction

After you trade an asset, the brokerage needs to settle the funds in your account. The most common type of settlement is immediate, where the funds are immediately available for trading after the transaction. In some cases, you may have delayed settlement, where you sell an asset and need to wait a few days to spend the cash you receive from the sale. A settlement model simulates these settlement rules.

Set Models

The brokerage model of your algorithm automatically sets the settlement model for each security, but you can override it. To manually set the settlement model of a security, call the `SetSettlementModel` method on the `Security` object.

C#

```
public override void Initialize()
{
    var security = AddEquity("SPY");
    // Set a delayed settlement model that settles 7 days after the trade at 8 AM
    // This can better mimic actual settlement of some brokerage, providing more realistic fund and margin available
    security.SetSettlementModel(new DelayedSettlementModel(7, TimeSpan.FromHours(8)));
}
```

You can also set the settlement model in a security initializer . If your algorithm has a universe, use the security initializer technique. In order to initialize single security subscriptions with the security initializer, call `AddSecurityInitializer` before you create the subscriptions.

C#

```
public class AddSecurityInitializerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the security initializer to set models of assets.
        AddSecurityInitializer(CustomSecurityInitializer);
    }

    private void CustomSecurityInitializer(Security security)
    {
        // Overwrite some of the reality models
        security.SetSettlementModel(new DelayedSettlementModel(7, TimeSpan.FromHours(8)));
    }
}
```

To view all the pre-built settlement models, see Supported Models .

Default Behavior

The brokerage model of your algorithm automatically sets the settlement model for each security. The default brokerage model is the `DefaultBrokerageModel` , which sets the settlement model based on the security type and your account type. The following table shows how it sets the settlement models:

Security Type	Account Type	Settlement Model
Equity	Cash	DelayedSettlementModel with the default settlement rules
Option	Cash	DelayedSettlementModel with the default settlement rules
Future	Any	FutureSettlementModel

For all other cases, the `DefaultBrokerageModel` uses the ImmediateSettlementModel .

The default delayed settlement rule for US Equity trades is T+2 at 8 AM Eastern Time (ET). For example, if you sell on Monday, the trade settles on Wednesday at 8 AM. The default delayed settlement rule for Future and Option contracts is T+1 at 8 AM.

Model Structure

Settlement models must extend the `ISettlementModel` interface. Extensions of the `ISettlementModel` interface must implement the `Scan` and `ApplyFunds` methods. The `Scan` method is automatically called at the top of each hour and it receives a `ScanSettlementModelParameters` object. The `ApplyFunds` method receives an `ApplyFundsSettlementModelParameters` object and applies the settlement rules. The `ApplyFunds` method is also automatically called, but its frequency depends on the security type. The `ApplyFunds` method is automatically called when you fill an order for the following security types:

- Equity
- Equity Options
- Crypto
- Forex
- Future Options
- Index Options

The `ApplyFunds` method is automatically called when you close a position for the following security types:

- Futures
- Crypto Futures
- CFD

```C#
public class CustomSettlementModelExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        var security = AddEquity("SPY");
        // Set the custom settlement model of the selected security to reflect actual scenario for more realistic
margin available
        security.SetSettlementModel(new MySettlementModel());
    }
}

// Define the custom settlement model outside of the algorithm
public class MySettlementModel : ISettlementModel
{
    public void ApplyFunds(ApplyFundsSettlementModelParameters applyFundsParameters)
    {
        var currency = applyFundsParameters.CashAmount.Currency;
        var amount = applyFundsParameters.CashAmount.Amount;
        applyFundsParameters.Portfolio.CashBook[currency].AddAmount(amount);
    }

    public void Scan(ScanSettlementModelParameters settlementParameters)
    {
    }

    public CashAmount GetUnsettledCash()
    {
        return new CashAmount(0, "USD");
    }
}
```

`ApplyFundsSettlementModelParameters` objects have the following properties:

`ScanSettlementModelParameters` objects have the following properties:

You likely don't need to create a custom settlement model because the supported models already implement immediate and delayed settlement rules.

Examples

The following examples demonstrate some common practices for implementing the settlement model.

Example 1: 2-day Settlement

The following algorithm simulates Interactive Brokers cash account. According to IB's website, cash positions will be settled in 48 hours. Thus, we can implement `DelayedSettlementModel` to mimic that. We hold the intra-day position of SPY to implement the settlement logic.

C#

```
public class SettlementModelAlgorithm : QCAlgorithm
{
    private Symbol _spy;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Simulate a cash account of IB.
        SetBrokerageModel(BrokerageName.InteractiveBrokersBrokerage, AccountType.Cash);
        // We need to set the settlement model in the security initializer.
        SetSecurityInitializer(new CustomSecurityInitializer(BrokerageModel, SecuritySeeder.Null));

        // Request SPY data for trading.
        _spy = AddEquity("SPY").Symbol;
    }

    public override void OnData(Slice slice)
    {
        // Trade based on updated data.
        if (slice.QuoteBars.TryGetValue(_spy, out var bar) && !Portfolio.Invested)
        {
            // Order based on the current available cash.
            var quantity = Math.Floor((Portfolio.Cash - Portfolio.UnsettledCash) / bar.Ask.Close);
            if (quantity > 0)
            {
                MarketOrder(_spy, quantity);
            }
        }

        // Exit position before market close; we only hold the intraday position.
        if (slice.Time.Hour == 15 && slice.Time.Minute == 45)
        {
            Liquidate();
        }
    }
}

class CustomSecurityInitializer : BrokerageModelSecurityInitializer
{
    public CustomSecurityInitializer(IBrokerageModel brokerageModel, ISecuritySeeder securitySeeder)
        : base(brokerageModel, securitySeeder)
    {

    }
    public override void Initialize(Security security)
    {
        base.Initialize(security);
        // IB settles cash in 48 hours.
        security.SetSettlementModel(new DelayedSettlementModel(2, new TimeSpan(0, 0, 0)));
    }
}
```

Example 2: No Unsettled Cash

The following algorithm implements a custom settlement model that states there is always $0 of unsettled cash.

C#

```
public class CustomSettlementModelAlgorithm : QCAlgorithm
{
    private bool _traded;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var security = AddEquity("SPY");
        security.SetSettlementModel(new MySettlementModel());

        Schedule.On(
            DateRules.EveryDay("SPY"),
            TimeRules.Every(TimeSpan.FromMinutes(30)),
            plotCash
        );

        _traded = false;
    }

    private void plotCash()
    {
        Plot("Settled Cash", "USD", Portfolio.CashBook["USD"].Amount);
        Plot("Unsettled Cash", "USD", Portfolio.UnsettledCashBook["USD"].Amount);
    }

    public override void OnData(Slice data)
    {
        if (_traded)
            return;

        MarketOrder("SPY", 1);
        MarketOrder("SPY", -1);
        _traded = true;
    }
}

public class MySettlementModel : ISettlementModel
{
    public void ApplyFunds(ApplyFundsSettlementModelParameters applyFundsParameters)
    {
        var currency = applyFundsParameters.CashAmount.Currency;
        var amount = applyFundsParameters.CashAmount.Amount;
        applyFundsParameters.Portfolio.CashBook[currency].AddAmount(amount);
    }

    public void Scan(ScanSettlementModelParameters settlementParameters)
    {
    }

    public CashAmount GetUnsettledCash()
    {
        return new CashAmount(0, "USD");
    }
}
```

Settlement

Supported Models

Introduction

This page describes all of the pre-built settlement models in LEAN. For more brokerage-specific settlement models, see the brokerage model documentation . If none of these models perform exactly how you want, create a custom settlement model .

Immediate Model

The `ImmediateSettlementModel` immediately adds or removes the cash from your portfolio when your transactions fill.

```
C#
security.SetSettlementModel(new ImmediateSettlementModel());
```

For more information about this model, see the class reference and implementation .

Delayed Model

The `DelayedSettlementModel` immediately removes the cash from your portfolio when your buy orders fill. When your sell orders fill, it adds the cash to your unsettled cash book . When the settlement period ends, the unsettled cash is added to your portfolio.

```
C#
security.SetSettlementModel(new DelayedSettlementModel(7, TimeSpan.FromHours(8)));
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`numberOfDays`	`int`	The number of days required for settlement	
`timeOfDay`	`TimeSpan`	The time of day used for settlement	

For more information about this model, see the class reference and implementation .

Account Currency Immediate Model

The `AccountCurrencyImmediateSettlementModel` applies cash settlement immediately and automatically converts the settlement cash into the account currency.

```
C#
security.SetSettlementModel(new AccountCurrencyImmediateSettlementModel());
```

For more information about this model, see the class reference and implementation .

Future Model

The `FutureSettlementModel` settles the daily profit and loss at the start of each day.

```C#
security.SetSettlementModel(new FutureSettlementModel());
```

For more information about this model, see the class reference and implementation .

Reality Modeling

Options Models

Options Models

Pricing

Introduction

Option price models compute the theoretical price of Option contracts, their implied volatility (IV), and their Greek values. Theoretical prices can help you detect undervalued and overvalued contracts, IV can provide you insight into the upcoming volatility of the underlying security, and Greek values can help you hedge your portfolio.

Data Discrepancies

There are several ways for you to get the Greeks and IV values, but you may notice some discrepancies between them.

Greeks in Universe Selection

The Greeks and IV values in the universe filter function are the daily, pre-calculated values based on the end of the previous trading day. You can't customize the Greeks and IV values that the filter function receives.

Greeks in Chain Requests

The Greeks and IV values in the `OptionUniverse` objects that you get from calling the `OptionChain` method are the daily, pre-calculated values based on the end of the previous trading day. You can't customize the Greeks and IV values that you get from this method.

Greeks in Data Events

The Greeks and IV values in the Slice that you get in the `OnData` method are the values from the Option price model. To customize the Greeks and IV values in the `Slice` , set a different price model.

Set Models

To set the pricing model of an Option, set its `PriceModel` property.

If you have access to the `Option` object when you subscribe to the Option universe or contract, you can set the price model immediately after you create the subscription.

```C#
public override Initialize()
{
    UniverseSettings.Asynchronous = true;
    var option = AddOption("SPY");
    // SPY options are American Options, in which IV and greeks more accurately calculated using Binomial model
    option.PriceModel = OptionPriceModels.BinomialCoxRossRubinstein();
}
```

Otherwise, set the price model in a security initializer .

```C#
public class AddSecurityInitializerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the security initializer to set models of assets.
        AddSecurityInitializer(CustomSecurityInitializer);
    }

    private void CustomSecurityInitializer(Security security)
    {
        // Overwrite the price model
        if (security.Type == SecurityType.Option) // Option type
        {
            (security as Option).PriceModel = OptionPriceModels.BinomialCoxRossRubinstein();
        }
    }
}
```

Default Behavior

The default Option pricing model is the `BinomialCoxRossRubinstein` for American Options or `BlackScholes` for European Options.

Disable Pricing

To turn off the Option price model, use the `CurrentPriceOptionPriceModel` . This model sets the Greeks to 0, sets the IV to 0, and sets the theoretical price to the current price.

```C#
option.PriceModel = new CurrentPriceOptionPriceModel();
```

Supported Models

LEAN supports the following Option price models.

Model	American Style	European Style
`AdditiveEquiprobabilities`	✓	✓
`BaroneAdesiWhaley`	✓	
`BinomialCoxRossRubinstein`	✓	✓
`BinomialJarrowRudd`	✓	✓
`BinomialJoshi`	✓	✓
`BinomialLeisenReimer`	✓	✓
`BinomialTian`	✓	✓
`BinomialTrigeorgis`	✓	✓
`BjerksundStensland`	✓	
`BlackScholes`		✓
`CrankNicolsonFD`	✓	✓
`Integral`		✓

If you set the price model of an Option to a model with an incompatible style, LEAN throws an exception.

What are Greeks?

Option Greeks measure the exposure of Option price or Option delta to the movement of different factors such as the underlying price, time, and volatility. The Greeks are a function of implied volatility. For more information about them, see The Greek Letters .

LEAN also provides indicator implementations of Option Greeks. It provides higher flexibility on Option price model selection, volatility modeling , and allows IV smoothing through call-put pair. For details, see Option Indicators .

What Is Implied Volatility?

Implied volatility is the forecasted future volatility of a security. The Option price model uses the realized volatility to calculate the implied volatility. By default, it uses the formula from Brenner and Subrahmanyam (1988) as the initial guess for implied volatility.

$$\frac{P}{S} \sqrt{\frac{2\pi}{T}}$$

where P is the Option contract price, S is the underlying price, and T is the time until Option expiration.

If the volatility model of the underlying security is ready , the price model uses its value as the initial guess for implied volatility and as an input to calculate the theoretical contract prices.

LEAN also provides an indicator implementation of implied volatility. It provides higher flexibility on Option price model selection, volatility modeling , and allows IV smoothing through call-put pair. For details, see Implied Volatility .

What Is Volatility Smile?

The "volatility smile" is a term used in finance to describe a particular shape that can be observed when plotting IV against strike

prices for options that share the same expiration date. Typically, when you plot IVs against strike prices, the curve tends to be concave, resembling a smile. This means that options with strike prices significantly above or below the current market price of the underlying asset tend to have higher IV than options with strike prices closer to the current market price.

Several theories attempt to explain the existence of the volatility smile. One explanation is that it arises from adjustments to the classic Black-Scholes-Merton (BSM) model, which assumes that asset prices follow a log-normal distribution. The volatility smile suggests that the market anticipates a higher probability of extreme price movements (fat tails) than the BSM model predicts. Another explanation is related to market dynamics and the demand for hedging. During periods of heightened market volatility, the volatility smile tends to become steeper, indicating a greater disparity in implied volatilities across different strike prices. This means that options, particularly those with out-of-the-money strike prices, become more expensive as investors seek protection against potential market downturns or increased price fluctuations. As a result, the volatility smile can serve as a valuable tool for investors and traders to assess market sentiment and gauge the level of risk in their portfolios.

According to some volatility models, such as the Heston model, the IV smile/surface should exhibit a smooth pattern. Any deviations from this smoothness, such as non-smooth spikes or irregularities, may signal potential arbitrage opportunities for traders.

Volatility Smirk

However, in some cases, rather than a traditional U-shaped curve, traders may observe what's known as a "smirk." This smirk occurs when the implied volatilities for higher-strike options are lower than those for ATM and lower-strike options. This deviation from the typical volatility smile pattern indicates a market expectation of a significant downward movement in the underlying asset's price. Traders interpret this as a signal that investors are allocating funds towards options with higher probabilities of such downward movements, resulting in increased demand for lower-strike options.

What Is Volatility Smoothing?

Volatility smoothing refers to the creation of a smooth volatility smile/surface. Derivative like options are favorable for maket-making and arbitration trading, due to the existence of fair price modeling. In the contemporary domain of high-frequency trading, the efficiency of calculating IV and Greeks assumes paramount significance. Thus, the quest for swift and refined IV surface computation becomes an inevitable quest for traders. Rather than relying solely on conventional methods involving slow-paced sampling and continuous parameter recalibration, the adoption of approximation techniques and stationary state analyses gains traction, especially in the context of IV estimation.

Accuracy Consideration

The Law of One Price stipulates the existence of a singular fair price for any given underlying asset. Consequently, according to the principles of put-call parity, each strike price for European options at expiration should correspond to a unique IV value. Within the realm of the Black-Scholes-Merton (BSM) framework, for instance, the concept of put-call parity yields insights into the equivalence of IVs for call and put options at identical strike prices and expirations. This equivalence serves as the foundation for formulating objective function: $C(t,K) - P(t,K)=e^{(q-r)t} (S_T - K) \\\Rightarrow e^{(q-r)t} [S_T (\Phi(d_{+}^{call})+\Phi(-d_{+}^{put})) - K (\Phi(d_{-}^{call})+\Phi(-d_{-}^{put}))] = e^{(q-r)t} (S_T - K) \\\Rightarrow \Phi(d_{+}^{call})+\Phi(-d_{+}^{put}) = 1\ \text{and}\ \Phi(d_{-}^{call})+\Phi(-d_{-}^{put}) = 1 \\\Rightarrow d_{\pm}^{call}=d_{\pm}^{put}$ Thus, we can formulate an objective function of $C_{market} + P_{market} = C_{BSM} + P_{BSM}$.

Alternatively, certain schools of thought contend that the bulk of historical volatility information is encapsulated within at-the-money (ATM) options, with any residual volatility attributable to the extrinsic value of the option. Consequently, a strategy focused on out-of-the-money (OTM) options for IV calculations gains traction.

Speed Consideration

In the realm of high-frequency trading, where split-second decisions are the norm, the traditional process of sampling, evaluating, recalibrating, and deriving implied volatility expectations is simply unfeasible. In response, traders and analysts increasingly turn to polynomial regression techniques as a pragmatic solution for IV modeling.

For example, $IV_t = \beta_0 + \beta_1 M + \beta_2 M^2 + \beta_3 (T-t) + \beta_4 (T-t)^2 + \beta_5 M(T-t) \\\text{where} M = ln(S_t/K)$

Additionally, the utilization of advanced mathematical algorithms, such as the Fast Fourier Transform (FFT), enables the rapid computation of the entire Greek surface, enhancing speed and efficiency. While these approaches may entail minor trade-offs in terms of accuracy, the overarching objective remains the attainment of a seamlessly smooth implied volatility surface that fosters effective arbitrage strategies.

Examples

Demonstration Algorithms
BasicTemplateMultiAssetAlgorithm.cs C# BasicTemplateOptionsHistoryAlgorithm.cs C#

Options Models

Volatility

Volatility

Key Concepts

Introduction

Volatility models measure the historical volatility of an asset. They are mostly used to calculate the volatility of the underlying security of an Option because the implied volatility of an Option contract needs an initial guess. The historical volatility doesn't need to be the standard deviation of the asset prices. The various volatility models in LEAN each have a unique methodology to calculate volatility.

LEAN also provides an indicator implementation of implied volatility. It provides higher flexibility on Option price model selection, volatility modeling , and allows IV smoothing through call-put pair. For details, see Implied Volatility .

Set Models

To set the volatility model of the underlying security of an Option, set the `VolatilityModel` property of the `Security` object. The volatility model can have a different resolution than the underlying asset subscription.

```C#
public override void Initialize()
{
    var underlyingSecurity = AddEquity("SPY");
    // Usually, the historical and implied volatility is calculated using 30-day standard deviation of return
    underlyingSecurity.VolatilityModel = new StandardDeviationOfReturnsVolatilityModel(30);
}
```

You can also set the volatility model in a security initializer . If your algorithm has a universe of underlying assets, use the security initializer technique. In order to initialize single security subscriptions with the security initializer, call `AddSecurityInitializer` before you create the subscriptions.

```C#
public class AddSecurityInitializerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the security initializer to set models of assets.
        AddSecurityInitializer(CustomSecurityInitializer);
    }

    private void CustomSecurityInitializer(Security security)
    {
        // Overwrite the volatility model
        if (security.Type == SecurityType.Equity)
        {
            security.VolatilityModel = new StandardDeviationOfReturnsVolatilityModel(30);
        }
    }
}
```

To view all the pre-built volatility models, see Supported Models .

Default Behavior

The default underlying volatility model for Equity Options and Index Options is the StandardDeviationOfReturnsVolatilityModel based on 30 days of daily resolution data. The default underlying volatility model for Future Options is the NullVolatilityModel .

Model Structure

Volatility models should extend the `BaseVolatilityModel` class. Extensions of the `BaseVolatilityModel` class must have `Update` and `GetHistoryRequirements` methods and a `Volatility` property. The `Update` method receives `Security` and `BaseData` objects and then updates the `Volatility` . The `GetHistoryRequirements` method receives `Security` and `DateTime` objects and then returns a list of `HistoryRequest` objects that represent the history requests to warm up the model. Volatility models receive data at each time step in the algorithm to update their state.

```C#
public class CustomVolatilityModelExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        var underlyingSecurity = AddIndex("VIX");
        // Apply custom volatility model for specific underlyings, often they are synthetic products or with
different market calendar
        underlyingSecurity.VolatilityModel = new MyVolatilityModel();
    }
}

// Define the custom volatility model outside of the algorithm
public class MyVolatilityModel : BaseVolatilityModel
{
    public override decimal Volatility { get; }

    public override void SetSubscriptionDataConfigProvider(
        ISubscriptionDataConfigProvider subscriptionDataConfigProvider)
    {
        SubscriptionDataConfigProvider = subscriptionDataConfigProvider;
    }

    public override void Update(Security security, BaseData data)
    {
    }

    public override IEnumerable<HistoryRequest> GetHistoryRequirements(
        Security security,
        DateTime utcTime)
    {
        return base.GetHistoryRequirements(security, utcTime);
    }

    public new IEnumerable<HistoryRequest> GetHistoryRequirements(
        Security security,
        DateTime utcTime,
        Resolution? resolution,
        int barCount)
    {
        return base.GetHistoryRequirements(security, utcTime, resolution, barCount);
    }
}
```

Warm Up Models

To use your volatility model as the inital guess for the implied volatility , warm up the volatility model of the underlying security. If you subscribe to all the Options in the `Initialize` method, set a warm-up period to warm up their volatility models. The warm-up period should provide the volatility models with enough data to compute their values.

```C#
public override void Initialize()
{
    // For default 30-day SD of return as volatility, you need 30+1 trading day to warm up
    SetWarmUp(31, Resolution.Daily);
}

public override void OnData(Slice slice)
{
    // Only take the warmed-up volatility value into account for accuracy issue
    if (IsWarmingUp) return;
}
```

If you have a dynamic universe of underlying assets and add Option contracts to your algorithm with the `AddOptionContract` , `AddIndexOptionContract` , or `AddFutureOptionContract` methods, warm up the volatility model when the underlying asset enters your universe. We recommend you do this inside a security initializer .

```C#
public class AddSecurityInitializerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the security initializer to set models of assets.
        AddSecurityInitializer(CustomSecurityInitializer);
    }

    private void CustomSecurityInitializer(Security security)
    {
        // Overwrite and warm up the volatility model
        if (security.Type == SecurityType.Equity) // Underlying asset type
        {
            security.VolatilityModel = new StandardDeviationOfReturnsVolatilityModel(30);
            foreach (var tradeBar in History(security.Symbol, 30, Resolution.Daily))
            {
                security.VolatilityModel.Update(security, tradeBar);
            }
        }
    }
}
```

Examples

The following examples demonstrate common practices for implementing a custom volatility model.

Example 1: Default Model Wrapper

The following algorithm defines a subclass of the `BaseVolatilityModel` and overrides its built-in members so you can easily customize the parts you need.

C#

```
public class CustomVolatilityModelAlgorithm : QCAlgorithm
{

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        var security = AddEquity("SPY", Resolution.Daily);
        security.VolatilityModel = new MyVolatilityModel();
    }

    /// OnData event is the primary entry point for your algorithm. Each new data point will be pumped in here.
    /// Slice object keyed by symbol containing the stock data
    public override void OnData(Slice data)
    {
        if (!Portfolio.Invested)
        {
            SetHoldings("SPY", 1);
        }
    }

}

public class MyVolatilityModel : BaseVolatilityModel
{
    public override decimal Volatility { get; }

    public override void SetSubscriptionDataConfigProvider(
        ISubscriptionDataConfigProvider subscriptionDataConfigProvider)
    {
        SubscriptionDataConfigProvider = subscriptionDataConfigProvider;
    }

    public override void Update(Security security, BaseData data)
    {
    }

    public override IEnumerable<HistoryRequest> GetHistoryRequirements(
        Security security,
        DateTime utcTime)
    {
        return base.GetHistoryRequirements(security, utcTime);
    }

    public new IEnumerable<HistoryRequest> GetHistoryRequirements(
        Security security,
        DateTime utcTime,
        Resolution? resolution,
        int barCount)
    {
        return base.GetHistoryRequirements(security, utcTime, resolution, barCount);
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
CustomVolatilityModelAlgorithm.cs C#

Volatility

Supported Models

Introduction

This page describes all of the pre-built volatility models in LEAN. If none of these models perform exactly how you want, create a custom volatility model .

Null Model

The `NullVolatilityModel` sets the volatility of the security to zero. It's the default volatility model for the underlying asset of Future Options.

```
C#
underlyingSecurity.VolatilityModel = VolatilityModel.Null;
```

For more information about this model, see the class reference and implementation .

Standard Deviation of Returns Model

The `StandardDeviationOfReturnsVolatilityModel` sets the volatility of the security to the annualized sample standard deviation of trailing returns. It's the default volatility model for the underlying asset of Equity Options and Index Options.

```
C#
underlyingSecurity.VolatilityModel = new StandardDeviationOfReturnsVolatilityModel(30);
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`periods`	`int`	The max number of samples to use when calculating the standard deviation of returns. This value must be greater than two.	
`resolution`	`Resolution?`	The resolution of the price data used to calculate the standard deviation. This only has a material effect in live mode. For backtesting, this value does not cause any behavioral changes.	`null`
`updateFrequency`	`TimeSpan?`	The frequency at which new values are inserted into the rolling window for the standard deviation calculation. If the value is `null`, it defaults to the `TimeSpan` representation of `resolution`. If the value and `resolution` are `null`, it defaults to a `TimeSpan` of one day.	`null`

For more information about this model, see the class reference and implementation.

Relative Standard Deviation Model

The `RelativeStandardDeviationVolatilityModel` sets the volatility of the security to the relative standard deviation of its price. In symbols, the value is

$$\frac{|\mu|}{\sigma}$$

where μ is the average of the samples and σ is the standard deviation of the samples.

```C#
underlyingSecurity.VolatilityModel = new RelativeStandardDeviationVolatilityModel(TimeSpan.FromDays(1), 10);
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`periodSpan`	`TimeSpan`	The period of time to wait between each sample of the security price	
`periods`	`int`	The number of samples to use to calculate the volatility value	

For more information about this model, see the class reference and implementation.

Options Models

Exercise

Introduction

If you exercise a long Option position or are assigned on your short Option position, LEAN processes an Option exercise order. The Option exercise model converts the Option exercise order into an OrderEvent .

Set Models

To set the exercise model of an Option, call the `SetOptionExerciseModel` method of the `Option` object inside a security initializer .

```C#
public class AddSecurityInitializerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the security initializer to set models of assets.
        AddSecurityInitializer(CustomSecurityInitializer);
    }

    private void CustomSecurityInitializer(Security security)
    {
        // Overwrite the Option exercise model
        if (security.Type == SecurityType.Option) // Option type
        {
            (security as Option).SetOptionExerciseModel(new DefaultExerciseModel());
        }
    }
}
```

Default Behavior

The default Option exercise model is the `DefaultExerciseModel` . The `DefaultExerciseModel` fills exercise orders to the full quantity with zero fees and applies an order tag to represent if the order is an exercise or assignment.

For more information about this model, see the class reference and implementation .

Model Structure

Option exercise models should implement the `IOptionExerciseModel` interface. The `IOptionExerciseModel` interface must implement the `OptionExercise` method, which receives `Option` and `OptionExerciseOrder` objects and then returns a list of `OrderEvent` objects that contain the order fill information.

```C#
public class CustomOptionExerciseModelExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        var security = AddOption("SPY");
        // Set custom option exercise model for mimicking specific Brokerage most realistic actions
        (security as Option).SetOptionExerciseModel(new MyOptionExerciseModel());
    }
}

// Define the custom Option exercise model outside of the algorithm
public class MyOptionExerciseModel : IOptionExerciseModel
{
    public IEnumerable<OrderEvent> OptionExercise(Option option, OptionExerciseOrder order)
    {
        var inTheMoney = option.IsAutoExercised(option.Underlying.Close);
        var isAssignment = inTheMoney && option.Holdings.IsShort;

        yield return new OrderEvent(
            order.Id,
            option.Symbol,
            option.LocalTime.ConvertToUtc(option.Exchange.TimeZone),
            OrderStatus.Filled,
            Extensions.GetOrderDirection(order.Quantity),
            0.0m,
            order.Quantity,
            OrderFee.Zero,
            "Tag"
        ) { IsAssignment = isAssignment };
    }
}
```

`OptionExerciseOrder` objects have the following properties:

The following table describes the arguments of the `OrderEvent` constructor:

Argument Details
Argument: `orderId` Id of the parent order Data Type: `int` | Default Value: `-`
Argument: `symbol` Asset Symbol Data Type: `Symbol` | Default Value: `-`
Argument: `utcTime` Date/time of this event Data Type: `DateTime` | Default Value: `-`

Argument: `direction` The direction of the order. The `OrderDirection` enumeration has the following members: Data Type: `OrderDirection` | Default Value: `Hold`
Argument: `fillPrice` Fill price information if applicable Data Type: `decimal` | Default Value: `0`
Argument: `fillQuantity` Fill quantity Data Type: `decimal` | Default Value: `0`
Argument: `orderFee` The order fee. You can use `OrderFee.Zero` or create an `OrderFee` object with a custom fee. `new OrderFee(new CashAmount(0.5m, "USD"));` Data Type: `OrderFee` | Default Value: `-`
Argument: `message` Message from the exchange Data Type: `string` | Default Value: `""`

`OrderEvent` objects have the following attributes:

Examples

The following examples demonstrate some common practices for implementing a custom Option exercise model.

Example 1: Cash Settlement

The following algorithm trades GOOG 30-day expiring straddle. Yet, instead of settling with the underlying stock, some brokerages will settle with cash for ITM options. To simulate this behavior, we can create a custom option exercise model.

C#

```
using static QuantConnect.Extensions;
public class OptionExerciseModelAlgorithm : QCAlgorithm
{
    private Symbol _goog;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request GOOG option data for trading.
        var security = AddOption("GOOG");
        _goog = security.Symbol;
        // Filter for the 2 ATM contracts expiring in 30 days to form a straddle strategy.
        security.SetFilter((universe) => universe.IncludeWeeklys().Straddle(30));
        // Set custom option exercise model for disabling exercise through security initializer.
        Settings.SeedInitialPrices = true;
        AddSecurityInitializer(security =>
        {
            // Set the custom Option exercise model for Option securities
            if (security.Type == SecurityType.Option)
            {
                (security as Option).SetOptionExerciseModel(new MyOptionExerciseModel());
            }
        });
    }

    public override void OnData(Slice slice)
    {
        // Open position on updated option chain data.
        if (!Portfolio.Invested && slice.OptionChains.TryGetValue(_goog, out var chain))
        {
            // Only one strike and expiry for the straddle universe.
            var strike = chain.Min(x => x.Strike);
            var expiry = chain.Min(x => x.Expiry);
            // Open the straddle position.
            var optionStrategy = OptionStrategies.Straddle(_goog, strike, expiry);
            Buy(optionStrategy, 5);
        }
    }

    // Define the custom Option exercise model outside of the algorithm
    private class MyOptionExerciseModel : IOptionExerciseModel
    {
        public IEnumerable<OrderEvent> OptionExercise(Option option, OptionExerciseOrder order)
        {
            var underlying = option.Underlying;
            var utcTime = option.LocalTime.ConvertToUtc(option.Exchange.TimeZone);
            var inTheMoney = option.IsAutoExercised(underlying.Close);
            // Cash settle: using payoff.
            var payoff = option.GetIntrinsicValue(underlying.Close);

            // Only liquidate option positions, but do not add equity positions.
            yield return new OrderEvent(
                order.Id,
                option.Symbol,
                utcTime,
                OrderStatus.Filled,
                GetOrderDirection(order.Quantity),
                payoff,
                order.Quantity,
                OrderFee.Zero,
                "Option Settlement"
            )
            {
                IsInTheMoney = inTheMoney
            };
        }
    }
}
```

Example 2: Default Model Implementation

The following algorithm implements the `DefaultExerciseModel` so you can easily customize the parts you need.

C#

```
public class CustomExerciseModelAlgorithm : QCAlgorithm
{
    private Symbol _optionSymbol;
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        Settings.SeedInitialPrices = true;
        AddSecurityInitializer(security =>
        {
            // Set the custom Option exercise model for Option securities
            if (security.Type == SecurityType.Option)
            {
                (security as Option).SetOptionExerciseModel(new MyOptionExerciseModel(this));
            }
        });
        var option = AddOption("SPY");
        _optionSymbol = option.Symbol;
    }

    /// OnData event is the primary entry point for your algorithm. Each new data point will be pumped in here.
    /// Slice object keyed by symbol containing the stock data
    public override void OnData(Slice data)
    {
        if (Portfolio.Invested)
        {
            return;
        }
        if (data.OptionChains.TryGetValue(_optionSymbol, out var chain))
        {
            var contract = chain
                .Where(x => x.Right == OptionRight.Call)
                .OrderBy(x => x.Strike)
                .FirstOrDefault();

            MarketOrder(contract.Symbol, 1);
            ExerciseOption(contract.Symbol, 1);
            Quit();
        }
    }
}

public class MyOptionExerciseModel : IOptionExerciseModel
{
    private readonly QCAlgorithm _algorithm;
    public MyOptionExerciseModel(QCAlgorithm algorithm) => _algorithm = algorithm;

    public IEnumerable<OrderEvent> OptionExercise(Option option, OptionExerciseOrder order)
    {
        var underlying = option.Underlying;
        var utcTime = option.LocalTime.ConvertToUtc(option.Exchange.TimeZone);

        var inTheMoney = option.IsAutoExercised(underlying.Close);
        var isAssignment = inTheMoney && option.Holdings.IsShort;

        yield return new OrderEvent(
            order.Id,
            option.Symbol,
            utcTime,
            OrderStatus.Filled,
            GetOrderDirection(order.Quantity),
            0.0m,
            order.Quantity,
            OrderFee.Zero,
            Messages.DefaultExerciseModel.ContractHoldingsAdjustmentFillTag(inTheMoney, isAssignment, option)
        )
        {
            IsAssignment = isAssignment,
            IsInTheMoney = inTheMoney
        };

        if (inTheMoney && option.ExerciseSettlement == SettlementType.PhysicalDelivery)
        {
            var exerciseQuantity = option.GetExerciseQuantity(order.Quantity);

            yield return new OrderEvent(
                order.Id,
                underlying.Symbol,
                utcTime,
```

```
                    OrderStatus.Filled,
                    GetOrderDirection(exerciseQuantity),
                    option.StrikePrice,
                    exerciseQuantity,
                    OrderFee.Zero,
                    isAssignment ? Messages.DefaultExerciseModel.OptionAssignment :
Messages.DefaultExerciseModel.OptionExercise
                ) { IsInTheMoney = true };
            }
        }
    }
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
CustomOptionExerciseModelRegressionAlgorithm.cs C#

Options Models

Assignment

Introduction

If you sell an Option in a backtest, an assignment model can simulate an Option exercise order on behalf of the buyer and assign you to complete the requirements of the contract.

Set Models

To set the assignment model of an Option, call the `SetOptionAssignmentModel` method of the `Option` object inside a security initializer .

```
C#
public class AddSecurityInitializerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the security initializer to set models of assets.
        AddSecurityInitializer(CustomSecurityInitializer);
    }

    private void CustomSecurityInitializer(Security security)
    {
        // Overwrite the assignment model
        if (security.Type == SecurityType.Option) // Option type
        {
            (security as Option).SetOptionAssignmentModel(new DefaultOptionAssignmentModel());
        }
    }
}
```

Default Behavior

The default Option assignment model is the `DefaultOptionAssignmentModel` . The `DefaultOptionAssignmentModel` scans your portfolio every hour. It considers exercising American-style Options if they are within 4 days of their expiration and it considers exercising European-style Options on their day of expiration. If you have sold an Option that's 5% in-the-money and the Option exercise order is profitable after the cost of fees, this model exercises the Option.

For more information about this model, see the class reference and implementation .

Model Structure

Option assignment models should implement the `IOptionAssignmentModel` interface. Extensions of the `IOptionAssignmentModel` interface must implement the `GetAssignment` method, which automatically fires at the top of each hour and returns the Option assignments to generate.

```C#
public class CustomOptionAssignmentModelExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        var security = AddOption("SPY");
        // Set custom option assignment model for mimicking specific Brokerage most realistic actions
        (security as Option).SetOptionAssignmentModel(new MyOptionAssignmentModel());
    }
}

// Define the custom Option assignment model outside of the algorithm
public class MyOptionAssignmentModel : IOptionAssignmentModel
{
    public OptionAssignmentResult GetAssignment(OptionAssignmentParameters parameters)
    {
        var option = parameters.Option;
        // Check if the contract is ITM
        if ((option.Right == OptionRight.Call && option.Underlying.Price > option.StrikePrice) ||
            (option.Right == OptionRight.Put && option.Underlying.Price < option.StrikePrice))
        {
            return new OptionAssignmentResult(option.Holdings.AbsoluteQuantity, "MyTag");
        }
        return OptionAssignmentResult.Null;
    }
}
```

The `OptionAssignmentParameters` object has the following members:

To exercise the Option, return an `OptionAssignmentResult` with a positive quantity. Otherwise, return `OptionAssignmentResult.Null`. The `OptionAssignmentResult` constructor accepts the following arguments:

Argument	Data Type	Description	Default Value
`quantity`	`decimal`	The quantity to assign	
`tag`	`string`	The order tag to use	

Disable Assignments

To disable early Option assignments, set the Option assignment model to the `NullOptionAssignmentModel`.

```C#
(security as Option).SetOptionAssignmentModel(new NullOptionAssignmentModel());
```

Examples

The following examples demonstrate some common practices for handling Option assignment.

Example 1: Liquidate After Assignment

The following algorithm trades a short straddle strategy on ATM SPY Options that expire within seven days. To capitalize the cash profit, it liquidates the underlying assigned Equity position in the `OnAssignmentOrderEvent` `on_assignment_order_event` method.

```C#
public class OptionAssignmentAlgorithm : QCAlgorithm
{
    private Option _option;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Add SPY Options data for trading.
        _option = AddOption("SPY");
        // Filter for the 2 ATM contracts that expire in 7 days to form a straddle.
        _option.SetFilter((universe) => universe.IncludeWeeklys().Straddle(7));
    }

    public override void OnData(Slice slice)
    {
        // Get the current Option chain and open a new position if we're not already invested.
        if (!Portfolio.Invested && slice.OptionChains.TryGetValue(_option.Symbol, out var chain))
        {
            // Select the strike and expiry of the contracts.
            var strike = chain.Min(x => x.Strike);
            var expiry = chain.Min(x => x.Expiry);
            // Open the straddle position.
            var optionStrategy = OptionStrategies.ShortStraddle(_option.Symbol, strike, expiry);
            Buy(optionStrategy, 1);
        }
    }

    public override void OnAssignmentOrderEvent(OrderEvent assignmentEvent)
    {
        // Liquidate the assigned SPY position.
        MarketOrder(
            assignmentEvent.Symbol.Underlying,
            -assignmentEvent.Quantity * _option.SymbolProperties.ContractMultiplier
        );
    }
}
```

Example 2: Wheel Strategy

The following algorithm demonstrates a wheel Option strategy . The strategy is split into separate parts to generate consistent income: selling puts and selling covered call.

C#

```
public class WheelStrategyAlgorithm : QCAlgorithm
{
    // Define the OTM threshold. A greater value means lower risk of assignment but lower premiums to collect.
    private readonly decimal _otmThreshold = 0.05m;
    private Equity _equity;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);

        // Seed the initial price of the each new security so we can add and trade contracts in the same time step.
        Settings.SeedInitialPrices = true;
        // Add SPY data for trading. Use raw data normalization mode to enable fair strike price comparison.
        _equity = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw);
    }

    private Symbol GetTargetContract(OptionRight right, decimal targetPrice)
    {
        // Get the Option chain.
        var chain = OptionChain(_equity.Symbol);
        // Select the contracts that are first to expire after 30 days since they have greater premiums and theta
        // to collect.
        var expiry = chain.Where(x => x.Expiry > Time.AddDays(30))
            .Min(x => x.Expiry);
        // Select the OTM Option contract to start the wheel or covered call.
        Symbol symbol;
        if (right == OptionRight.Call)
        {
            symbol = chain.Where(x => x.Expiry == expiry && x.Right == right && x.Strike >= targetPrice)
                .OrderBy(x => x.Strike)
                .First();
        }
        else
        {
            symbol = chain.Where(x => x.Expiry == expiry && x.Right == right && x.Strike <= targetPrice)
                .OrderByDescending(x => x.Strike)
                .First();
        }
        // Add the Option contract to the algorithm.
        return AddOptionContract(symbol).Symbol;
    }

    public override void OnData(Slice slice)
    {
        // Start the wheel by selling the OTM put.
        if (!Portfolio.Invested && IsMarketOpen(_equity.Symbol))
        {
            var symbol = GetTargetContract(OptionRight.Put, _equity.Price * (1m - _otmThreshold));
            SetHoldings(symbol, -0.2m);
        }
    }

    public override void OnAssignmentOrderEvent(OrderEvent assignmentEvent)
    {
        // After the put is assigned, look for an OTM call to form a covered call with the assigned underlying
        // position.
        var symbol = GetTargetContract(OptionRight.Call, _equity.Price * (1m + _otmThreshold));
        MarketOrder(symbol, -_equity.Holdings.Quantity / 100m);
    }
}
```

Example 3: Assign ITM Contracts

The following algorithm assigns all in-the-money contracts at the top of each hour.

C#

```
public class CustomOptionAssignmentAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        AddSecurityInitializer(InitializeSecurity);
        var option = AddOption("GOOG");
        option.SetFilter(universe => universe.CallsOnly().Strikes(-65, 0).Expiration(0, 30));
    }

    private void InitializeSecurity(Security security)
    {
        // Overwrite the assignment model.
        if (security.Type == SecurityType.Option) // Option type
        {
            (security as Option).SetOptionAssignmentModel(new MyOptionAssignmentModel());
        }
    }

    public override void OnData(Slice data)
    {
        if (Portfolio.Invested) return;

        foreach (var kvp in data.OptionChains)
        {
            var chain = kvp.Value;
            var minExpiry = chain.Select(x => x.Expiry).Min();
            var contracts = chain.Where(x => x.Expiry == minExpiry).OrderBy(x => x.Strike).ToList();

            if (contracts.Any())
            {
                MarketOrder(contracts[0].Symbol, -1);
            }
        }
    }
}

public class MyOptionAssignmentModel : IOptionAssignmentModel
{
    public OptionAssignmentResult GetAssignment(OptionAssignmentParameters parameters)
    {
        var option = parameters.Option;
        // Check if the contract is ITM
        if ((option.Right == OptionRight.Call && option.Underlying.Price > option.StrikePrice) ||
            (option.Right == OptionRight.Put && option.Underlying.Price < option.StrikePrice))
        {
            return new OptionAssignmentResult(option.Holdings.AbsoluteQuantity, "MyTag");
        }

        return OptionAssignmentResult.Null;
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
CustomOptionAssignmentRegressionAlgorithm.cs C#

Reality Modeling

Risk Free Interest Rate

Risk Free Interest Rate

Key Concepts

Introduction

The risk free interest rate is the hurdle rate for investments. If investors can get a rate of return with virtually no risk, an investment needs to offer the potential for a greater return to attract capital from rational investors. The risk free interest rate also impacts some performance statistics , like Sharpe and Sortino ratios. Risk free interest rate models provide the interest rate data to your algorithms to enable these calculations.

Set Models

To set the risk free interest rate model, in the Initialize method, call the `SetRiskFreeInterestRateModel` method.

```C#
public overide void Initialize()
{
    // Set the interest rate model for accurate option IV/greeks and result metrics like Sharpe Ratio
    // You should set it according to the main currency of your trading Securities
    SetRiskFreeInterestRateModel(new ConstantRiskFreeRateInterestRateModel(0.02m));
}
```

To view all the pre-built risk free interest rate models, see Supported Models .

Default Behavior

The default risk free interest rate model is the InterestRateProvider , which provides the primary credit rate from the Federal Open Market Committee (FOMC).

Model Structure

Risk free interest rate models must extend the `IRiskFreeInterestRateModel` interface. Extensions of the `IRiskFreeInterestRateModel` interface must implement a `GetInterestRate` method. The `GetInterestRate` method returns the risk free interest rate for a given date.

```C#
public class CustomRiskFreeInterestRateModelExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // Set the interest rate model for accurate option IV/greeks and result metrics like Sharpe Ratio
        // You should set it according to the main currency of your trading Securities
        SetRiskFreeInterestRateModel(new MyRiskFreeInterestRateModel());
    }
}

// Define the custom risk free interest rate model
public class MyRiskFreeInterestRateModel : IRiskFreeInterestRateModel
{
    public decimal GetInterestRate(DateTime date)
    {
        return 0.02m;
    }
}
```

Disable Interest Rates

To disable the risk free interest rate, set the model to the `ConstantRiskFreeRateInterestRateModel` and set the risk free rate to zero.

```C#
SetRiskFreeInterestRateModel(new ConstantRiskFreeRateInterestRateModel(0));
```

Get Interest Rate

To get the risk free interest rate for a specific time, call the `GetInterestRate` method with the time.

```C#
public override void OnData(Slice slice)
{
 var riskFreeInterestRate = RiskFreeInterestRateModel.GetInterestRate(slice.Time);
}
```

To get the average risk free interest rate for a window of time, call the `GetRiskFreeRate` method with the start date and end date.

```C#
public override void OnData(Slice slice)
{
 var avgRiskFreeInterestRate = RiskFreeInterestRateModel.GetRiskFreeRate(Time.AddDays(-30), Time);
}
```

To get the average risk free interest rate for a set of dates, call the `GetAverageRiskFreeRate` method with the list of dates.

```C#
public override void OnData(Slice slice)
{
 var avgRiskFreeInterestRate = RiskFreeInterestRateModel.GetAverageRiskFreeRate(
  new [] {Time, Time.AddDays(-180), Time.AddDays(-365)}
 );
}
```

Examples

The following examples demonstrate common practices for implementing a custom risk free interest rate model.

Example 1: Fixed Rate

The following algorithm implements a custom risk free interest rate model with a fixed rate of 2%.

```C#
public class CustomFillModelAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        SetRiskFreeInterestRateModel(new MyRiskFreeInterestRateModel());
        AddEquity("SPY", Resolution.Daily);
    }

    public override void OnData(Slice data)
    {
        if (!Portfolio.Invested)
        {
            SetHoldings("SPY", 1);
        }
    }
}

public class MyRiskFreeInterestRateModel : IRiskFreeInterestRateModel
{
    public decimal GetInterestRate(DateTime date)
    {
        return 0.02m;
    }
}
```

Risk Free Interest Rate

Supported Models

Introduction

This page describes all of the pre-built risk free interest rate models in LEAN. If none of these models perform exactly how you want, create a custom risk free interest rate model .

Constant Model

The `ConstantRiskFreeRateInterestRateModel` returns a constant rate across time. It's the default risk free interest rate model.

```
C#
SetRiskFreeInterestRateModel(new ConstantRiskFreeRateInterestRateModel(0.02m));
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`riskFreeRate`	`decimal`	The risk free interest rate	

For more information about this model, see the class reference and implementation .

Function Model

The `FuncRiskFreeRateInterestRateModel` calls a function you provide to get the risk free interest rate. The function you pass to the constructor effectively replaces the GetInterestRate method.

```
C#
SetRiskFreeInterestRateModel(new FuncRiskFreeRateInterestRateModel(getInterestRateFunc));
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`getInterestRateFunc`	`Func<DateTime, decimal>`	A function that returns the risk free interest rate for a given date	

For more information about this model, see the class reference and implementation .

Interest Rate Provider Model

The `InterestRateProvider` returns the primary credit rate from the Federal Open Market Committee (FOMC).

```
C#
SetRiskFreeInterestRateModel(new InterestRateProvider());
```

For more information about this model, see the class reference and implementation .

Reality Modeling

Dividend Yield

Dividend Yield

Key Concepts

Introduction

Dividends are periodic payments a company makes to its shareholders out of its earnings. The dividend yield refers to the rate of return an investor earns in the form of dividends from holding a stock.

In the context of Option pricing models , the dividend yield is a critical factor because it affects the value of the underlying stock. When calculating the theoretical price of an Option using these models, the dividend yield is factored into the equation.

Set Models

To set the dividend yield model for an Option indicator , pass a `DividendYieldProvider` as the `dividendYieldModel` parameter.

```C#
public override void Initialize()
{
    // Before creating the indicator, create the set dividend yield model, such that it will use it in default
    // Using DividendYieldProvider will calculate the continuous dividend yield using all dividend payoffs in 1 year
    var dividendYieldModel = new DividendYieldProvider(symbol.Underlying);
    _iv = new ImpliedVolatility(symbol, RiskFreeInterestRateModel, dividendYieldModel,
OptionPricingModelType.BlackScholes);
}
```

To view all the pre-built dividend yield models, see Supported Models .

Default Behavior

For US Equity Options, the default dividend yield model is the DividendYieldProvider , which provides the continuous yield calculated from all dividend payoffs from the underlying Equity over the previous 350 days.

For other Option types, the default dividend yield model is the `DividendYieldProvider` , but it calculates the yield from the dividends of the SPY ETF.

Model Structure

Dividend yield models must extend the `IDividendYieldModel` interface. Extensions of the `IDividendYieldModel` interface must implement a `GetDividendYield` method. The `GetDividendYield` method returns the dividend yield for a given date.

```C#
public class CustomDividendYieldModelExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Create the dividend yield model
        var dividendYieldModel = new MyDividendYieldModel();
        // Add the Option contracts.
        var underlying = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;
        var strike = 595m;
        var expiry = new DateTime(2024, 12, 30);
        var option = AddContract(underlying, OptionRight.Put, strike, expiry);
        var mirror = AddContract(underlying, OptionRight.Call, strike, expiry);
        // Add the indicator.
        var iv = new ImpliedVolatility(option, RiskFreeInterestRateModel, dividendYieldModel, mirror);
    }

    private Symbol AddContract(Symbol underlyingSymbol, OptionRight right, decimal strike, DateTime expiry)
    {
        var contractSymbol = QuantConnect.Symbol.CreateOption(underlyingSymbol, Market.USA, OptionStyle.American,
right, strike, expiry);
        AddOptionContract(contractSymbol);
        return contractSymbol;
    }
}

// Define the custom dividend yield model
public class MyDividendYieldModel : IDividendYieldModel
{
    decimal _yield = 0.02m;
    public decimal GetDividendYield(DateTime date)
    {
        return _yield;
    }
    public decimal GetDividendYield(DateTime date, decimal price)
    {
        return _yield;
    }
}
```

Examples

The following examples demonstrate common practices for implementing a custom dividend yield model.

Example 1: Fixed Yield

The following algorithm implements a custom dividend yield model with a fixed yield of 2%.

C#

```
public class CustomDividendYieldModelAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Dictionary<Symbol, Delta> _deltas = new();
    private Dictionary<Symbol, MyDividendYieldModel> _dividendYieldModel = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var option = AddOption("MSFT");
        option.SetFilter(0,1,0,31);
        _symbol = option.Symbol;
        _dividendYieldModel[_symbol.Underlying] = new MyDividendYieldModel();
    }

    public override void OnData(Slice data)
    {
        if (!data.OptionChains.TryGetValue(_symbol, out var chain)) return;
        var expiry = chain.Min(x => x.Expiry);
        chain
            .Where(x => x.Expiry == expiry && x.Right == OptionRight.Call && !_deltas.ContainsKey(x.Symbol))
            .Select(x => x.Symbol)
            .DoForEach(x => _deltas[x] = CreateDeltaIndicator(x));
    }

    private Delta CreateDeltaIndicator(Symbol symbol)
    {
        var dividendYieldModel = _dividendYieldModel[symbol.Underlying];
        var delta = new Delta(symbol.Value, symbol, RiskFreeInterestRateModel, dividendYieldModel);
        RegisterIndicator(symbol, delta, Resolution.Minute);
        RegisterIndicator(symbol.Underlying, delta, Resolution.Minute);
        return delta;
    }

    public override void OnEndOfDay(Symbol symbol)
    {
        if (_deltas.TryGetValue(symbol, out var delta))
        {
            Plot("Delta", symbol.Value, delta.Current.Value);
        }
    }
}

public class MyDividendYieldModel : IDividendYieldModel
{
    public decimal GetDividendYield(DateTime date)
    {
        return 0.02m;
    }
}
```

Dividend Yield

Supported Models

Introduction

This page describes all of the pre-built dividend yield models in LEAN. If none of these models perform exactly how you want, create a custom dividend yield model .

Constant Model

The `ConstantDividendYieldModel` returns a constant yield across time. It's the default dividend yield model for securities that don't pay dividends.

```
C#
_dividendYieldModel = new ConstantDividendYieldModel(0.02m);
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`dividendYield`	`decimal`	The dividend yield	

For more information about this model, see the class reference and implementation .

Dividend Yield Provider Model

The `DividendYieldProvider` uses historical dividend payments to estimate the annualized continuous dividend yield for a given date.

```
C#
_dividendYieldModel = new DividendYieldProvider();
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`symbol`	`Symbol`	The symbol of a dividend-paying security. If you don't provide a value, it defaults to SPY.	

For more information about this model, see the class reference and implementation .

Reality Modeling

Margin Interest Rate

Margin Interest Rate

Key Concepts

Introduction

Margin interest is a cost associated with trading on margin. Margin interest rate models model margin interest cash flows by directly adding or removing cash from your portfolio.

Set Models

The brokerage model of your algorithm automatically sets the margin interest rate model for each security, but you can override it. To manually set the margin interest rate model of a security, assign a model to the `MarginInterestRateModel` property of the Security object.

```
C#
public override void Initialize()
{
    var security = AddEquity("SPY");
    // For US Equity, use the ShortMarginInterestRateModel with IB data
    security.SetShortableProvider(new InteractiveBrokersShortableProvider());
    security.MarginInterestRateModel = new ShortMarginInterestRateModel();
}
```

You can also set the margin interest rate model in a security initializer . If your algorithm has a dynamic universe, use the security initializer technique. In order to initialize single security subscriptions with the security initializer, call `AddSecurityInitializer` before you create the subscriptions.

```
C#
public class AddSecurityInitializerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the security initializer to set models of assets.
        AddSecurityInitializer(CustomSecurityInitializer);
    }

    private void CustomSecurityInitializer(Security security)
    {
        // Overwrite some of the reality models
        security.SetShortableProvider(new InteractiveBrokersShortableProvider());
        security.MarginInterestRateModel = new ShortMarginInterestRateModel();

    }
}
```

To view all the pre-built margin interest rate models, see Supported Models .

Default Behavior

The brokerage model of your algorithm automatically sets the margin interest rate model of each security. The default brokerage model is the `DefaultBrokerageModel` , which sets the NullMarginInterestRateModel .

Model Structure

Margin interest rate models should implement the `IMarginInterestRateModel` interface. Extensions of the `IMarginInterestRateModel` interface must implement the `ApplyMarginInterestRate` method, which applies margin interest payments to the portfolio.

```
C#
public class CustomMarginInterestRateModelExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        var security = AddEquity("SPY");
        // You should set it according to your broker information for the least actual-expected discrepancies
        security.SetMarginInterestRateModel(new MyMarginInterestRateModel());
    }
}

// Define the custom margin interest rate model
public class MyMarginInterestRateModel : IMarginInterestRateModel
{
    public void ApplyMarginInterestRate(MarginInterestRateParameters marginInterestRateParameters)
    {
        var holdings = marginInterestRateParameters.Security.Holdings;
        var positionValue = holdings.GetQuantityValue(holdings.Quantity);
        positionValue.Cash.AddAmount(-1);
    }
}
```

The `ApplyMarginInterestRate` method is automatically called at the top of each hour.

Examples

The following examples demonstrate some common practices for implementing a custom margin interest rate model.

Example 1: Based On Interest Rate

The following algorithm buys and holds SPY, with 2x leverage and holds 1.5x of portfolio value. Hence, funds will be borrowed from the broker and are susceptible to margin interest. We implement a custom margin interest rate model to simulate the Interactive Brokers margin interest deduction. While following daily accrual and monthly accounting of the margin interest, we simplify the rate from tiered to fixed at 1% above the current interest rate.

```
C#
public class MarginInterestRateModelAlgorithm : QCAlgorithm
{
    private Symbol _spy;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);

        // Request SPY data for trading.
        var equity = AddEquity("SPY");
        _spy = equity.Symbol;
        // Set custom margin interest rate model for the SPY security to mimic IB margin interest deduction.
        equity.SetMarginInterestRateModel(new MyMarginInterestRateModel(this, _spy));
    }

    public override void OnData(Slice slice)
    {
        if (!Portfolio.Invested)
        {
            // Place a position that exceeds available cash to buy with borrowed funds.
            SetHoldings(_spy, 1.5m);
```

```
        }
    }

    // Define the custom margin interest rate model
    public class MyMarginInterestRateModel : IMarginInterestRateModel
    {
        private QCAlgorithm _algorithm;
        private Dictionary<Cash, decimal> _accruedMarginInterest = new();

        public MyMarginInterestRateModel(QCAlgorithm algorithm, Symbol symbol)
            : base()
        {
            // Save the algorithm instance since we need to access the at-the-time interest rate to calculate the
margin rate.
            _algorithm = algorithm;
            // IBKR accrues interest daily and posts actual interest monthly on the third business day of the
following month.
            algorithm.Schedule.On(
                algorithm.DateRules.MonthStart(symbol, 3),
                algorithm.TimeRules.At(0, 0),
                MarginDeduction
            );
        }

        private void MarginDeduction()
        {
            foreach (var (cash, accruedInterest) in _accruedMarginInterest)
            {
                // Monthly deduction of cash position with accrued interests.
                cash.AddAmount(accruedInterest);
                // Reset the accrued interest accumulator.
                _accruedMarginInterest[cash] = 0m;
            }
        }

        public void ApplyMarginInterestRate(MarginInterestRateParameters marginInterestRateParameters)
        {
            // IBKR accrues interest daily for overnight positions only.
            if (_algorithm.Time.Hour == 16)
            {
                var holdings = marginInterestRateParameters.Security.Holdings;
                // Calculate the borrowed capital. The margin interest will only applied to the borrowed capital.
                var borrowedValue = holdings.UnleveredAbsoluteHoldingsCost - holdings.AbsoluteHoldingsCost;
                // Margin interest rate is 1% above the current risk-free interest rate.
                var marginRate = _algorithm.RiskFreeInterestRateModel.GetInterestRate(_algorithm.Time) + 0.01m;
                // Deduct the margin interest on the account cash position by the margin rate, multiplying the
borrowed sum.
                var cash = holdings.GetQuantityValue(holdings.Quantity).Cash;
                _accruedMarginInterest.TryAdd(cash, 0m);
                // Daily rate will be divided by 365.
                _accruedMarginInterest[cash] += borrowedValue * marginRate / 365m;

            }
        }
    }
}
```

Example 2: Fixed Dollar Rate

The following algorithm defines custom margin interest rate model that reduces your cash by $1 at the top of each hour.

C#

```
public class CustomFillModelAlgorithm : QCAlgorithm
{

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        var security = AddEquity("SPY", Resolution.Daily);
        security.SetMarginInterestRateModel(new MyMarginInterestRateModel());
    }

    public override void OnData(Slice data)
    {
        if (!Portfolio.Invested)
        {
            SetHoldings("SPY", 1);
        }
    }
}

public class MyMarginInterestRateModel : IMarginInterestRateModel
{
    public void ApplyMarginInterestRate(MarginInterestRateParameters marginInterestRateParameters)
    {
        var holdings = marginInterestRateParameters.Security.Holdings;
        var positionValue = holdings.GetQuantityValue(holdings.Quantity);
        positionValue.Cash.AddAmount(-1);
    }
}
```

Margin Interest Rate

Supported Models

Introduction

This page describes the pre-built margin interest rate models in LEAN. If none of these models perform exactly how you want, create a custom margin interest rate model .

Null Model

The `NullMarginInterestRateModel` doesn't charge any margin interest. It's the default margin interest rate of the DefaultBrokerageModel .

```
C#
security.margin_interest_rate_model = MarginInterestRateModel.NULL;
```

For more information about this model, see the class reference and implementation .

Short Equity Model

The `ShortMarginInterestRateModel` simulates the margin cost and payments of your Equity holdings. This model use the fee rate to compute the amount due to borrowing the asset to short, and the rebate rate to compute the amount due to lending the asset for shorting. For more information about these rates providers, see Short Availability .

```
C#
security.SetShortableProvider(new InteractiveBrokersShortableProvider());
security.MarginInterestRateModel = new ShortMarginInterestRateModel();
```

For more information about this model, see the class reference and implementation .

Binance Futures Model

The `BinanceFutureMarginInterestRateModel` simulates the margin cost and payments of your Crypto Future holdings. When the funding rate is positive, the price of the perpetual contract is higher than the mark price, so traders who are long pay for short positions. Conversely, a negative funding rate indicates that perpetual prices are below the mark price, so traders who are short pay for long positions. The interest amount is

$$Nominal \ Value \ of \ Positions * Funding \ Rate$$

The interest amount is periodically credited or debited from your currency holdings while you hold open positions. The interest amount is charged at 12 AM, 8 AM, 4 PM in your algorithm time zone .

```
C#
security.MarginInterestRateModel = new BinanceFutureMarginInterestRateModel();
```

For more information about this model, see the class reference and implementation .

Bybit Futures Model

The `BybitFutureMarginInterestRateModel` has the same behavior as the BinanceFutureMarginInterestRateModel .

```
security.MarginInterestRateModel = new BybitFutureMarginInterestRateModel();
```

C#

For more information about this model, see the class reference and implementation .

Reality Modeling

Margin Calls

Introduction

If the value of your portfolio falls below the maintenance margin, you receive a margin call. If you receive a margin call, you either need to add more capital to your brokerage account or the brokerage will liquidate some of your holdings to reduce your exposure and their risk. A margin call model monitors the margin levels of your portfolio, issues margin call warnings, and submits orders when margin calls occur.

Set Models

To set the margin call model, call the `SetMarginCallModel` method of the `Portfolio` object.

```
C#
public override void Initialize()
{
 // Set margin call model for the whole portfolio, better with reference to your broker information
 Portfolio.SetMarginCallModel(new DefaultMarginCallModel(Portfolio, DefaultOrderProperties));
}
```

Default Behavior

The brokerage model of your algorithm automatically sets the margin call model for the portfolio. The default brokerage model is the `DefaultBrokerageModel` , which sets the `DefaultMarginCallModel` . The `DefaultMarginCallModel` issues margin call warnings when the margin remaining in your portfolio is less than or equal to 5% of the total portfolio value. When your total margin used exceeds 10% of the total portfolio value, this model creates margin call orders. It sorts the generated margin call orders in ascending order by their unrealized profit and then synchronously executes each order until your portfolio is within the margin requirements.

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`portfolio`	`SecurityPortfolioManager`	The portfolio object to receive margin calls.	
`defaultOrderProperties`	`IOrderProperties`	The default order properties to be use in margin call orders.	
`marginBuffer`	`decimal`	The percent margin buffer to use when checking whether the total margin used is above the total portfolio value to generate margin call orders.	`0.10m` (10%)

For more information about this model, see the class reference and implementation .

Model Structure

Margin call models must extend the `DefaultMarginCallModel` class. Extensions of the `DefaultMarginCallModel` class can override the `GetMarginCallOrders` and `ExecuteMarginCall` methods.

The `GetMarginCallOrders` method scans the portfolio and the updated data for a potential margin call situation that may get the equity below zero. The method must return a list of `SubmitOrderRequest` objects that represent the margin call orders. To issue a margin call warning during this method, set the `issueMarginCallWarning` argument of the method to true.

The `ExecuteMarginCall` method receives the list of `SubmitOrderRequest` objects from the `GetMarginCallOrders` method, executes some of them, and returns a list of `OrderTicket` objects.

```
C#
public class CustomMarginCallModelExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // Set margin call model for the whole portfolio, better with reference to your broker information
        Portfolio.SetMarginCallModel(new MyMarginCallModel(Portfolio, DefaultOrderProperties));
    }
}

// Define the custom margin call model
public class MyMarginCallModel : DefaultMarginCallModel
{
    public MyMarginCallModel(
        SecurityPortfolioManager portfolio,
        IOrderProperties defaultOrderProperties)
        : base(portfolio, defaultOrderProperties)
    {
    }

    public override List<OrderTicket> ExecuteMarginCall(
        IEnumerable<SubmitOrderRequest> generatedMarginCallOrders)
    {
        return base.ExecuteMarginCall(generatedMarginCallOrders);
    }

    public List<SubmitOrderRequest> GetMarginCallOrders(out bool issueMarginCallWarning)
    {
        return base.GetMarginCallOrders(out issueMarginCallWarning);
    }
}
```

Disable Margin Calls

To disable margin calls, set the margin call model to the `NullMarginCallModel` .

```
C#
public override void Initialize()
{
 Portfolio.MarginCallModel = MarginCallModel.Null;
}
```

Monitor Margin Call Events

When the margin call model of your portfolio issues a margin call warning, we notify your algorithm through the `OnMarginCallWarning` event handler.

```
C#
public override void OnMarginCallWarning()
{
    Debug("Warning: Close to margin call");
}
```

Before we send the orders that the margin call model produces, we notify your algorithm through the `OnMarginCall` event

handler. This notification gives your algorithm a chance to liquidate some positions or modify the margin call orders. To modify the orders, adjust the list of `SubmitOrderRequest` objects the event handler receives.

```C#
public override void OnMarginCall(List<SubmitOrderRequest> requests)
{
    for (var i = 0; i < requests.Count; i++)
    {
        var order = requests[i];
        // liquidate an extra 10% each time you get a margin call to give yourself more padding
        var newQuantity = (int)(order.Quantity * 1.1m);
        requests[i] = new SubmitOrderRequest(order.OrderType, order.SecurityType,
                                             order.Symbol, newQuantity, order.StopPrice,
                                             order.LimitPrice, 0, Time, "OnMarginCall");
    }
}
```

Submit Order Request

If you receive a margin call or create a margin call model, you'll need to work with `SubmitOrderRequest` objects. These objects have the following properties:

The following table describes the arguments of the `SubmitOrderRequest` constructor:

Argument	Data Type	Description	Default Value
`orderType`	`OrderType`	The order type to be submitted	
`securityType`	`SecurityType`	The security type of the asset	
`symbol`	`Symbol`	The symbol to be traded	
`quantity`	`decimal`	The number of units to order	
`stopPrice`	`decimal`	The stop price for stop orders	
`limitPrice`	`decimal`	The limit price for limit orders	
`triggerPrice`	`decimal`	The trigger price for limit if touched orders	
`time`	`DateTime`	The time this request was created	
`tag`	`string`	A custom tag for this request	
`properties`	`IOrderProperties`	The order properties for this request	`null`

Examples

The following examples demonstrate common practices for implementing a custom margin call model.

Example 1: Default Model Wrapper

The following algorithm defines a subclass of the `DefaultMarginCallModel` and overrides its built-in members so you can easily customize the parts you need.

```C#
public class CustomFillModelAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        Portfolio.MarginCallModel = new MyMarginCallModel(Portfolio, DefaultOrderProperties);
        AddEquity("SPY", Resolution.Daily);
    }

    public override void OnData(Slice data)
    {
        if (!Portfolio.Invested)
        {
            SetHoldings("SPY", -2);
        }
    }
}

public class MyMarginCallModel : DefaultMarginCallModel
{
    public MyMarginCallModel(
        SecurityPortfolioManager portfolio,
        IOrderProperties defaultOrderProperties)
        : base(portfolio, defaultOrderProperties)
    {
    }

    public override List<OrderTicket> ExecuteMarginCall(
        IEnumerable<SubmitOrderRequest> generatedMarginCallOrders)
    {
        return base.ExecuteMarginCall(generatedMarginCallOrders);
    }

    public List<SubmitOrderRequest> GetMarginCallOrders(out bool issueMarginCallWarning)
    {
        return base.GetMarginCallOrders(out issueMarginCallWarning);
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
MarginCallEventsAlgorithm.cs C#

Reality Modeling

Short Availability

Short Availability

Key Concepts

Introduction

To short a security, you need to borrow shares from another investor or organization that owns the shares. A shortable provider is a model that tracks the number of shares that are available for you to borrow. A shortable provider can make your backtest results more realistic because it doesn't let you place a short trade if there are no shares available for you to borrow.

Set Providers

The brokerage model of your algorithm automatically sets the settlement model for each security, but you can override it. To manually set the shortable provider of a security, call the `SetShortableProvider` method on the `Security` object.

C#

```
public override void Initialize()
{
    var security = AddEquity("SPY");
    // Get shortable stocks and quantity from local disk
    security.SetShortableProvider(new LocalDiskShortableProvider("axos"));
}
```

You can also set the shortable provider in a security initializer. If your algorithm has a universe, use the security initializer technique. In order to initialize single security subscriptions with the security initializer, call `AddSecurityInitializer` before you create the subscriptions.

C#

```
public override void Initialize()
{
    // Set the security initializer before requesting data to apply to all requested securities afterwards
    SetSecurityInitializer(CustomSecurityInitializer);
    AddEquity("SPY");
}

private void CustomSecurityInitializer(Security security)
{
    security.SetShortableProvider(new LocalDiskShortableProvider("axos"));
}
```

If you call the `SetSecurityInitializer` method, it overwrites the default security initializer. The default security initializer uses the security-level reality models of the brokerage model to set the following reality models of each security:

- Fill
- Slippage
- Fee

- Buying Power
- Settlement
- Short availability
- Margin Interest Rate

To extend upon the default security initializer instead of overwriting it, call `AddSecurityInitializer` before you create the subscriptions before you create the subscriptions.

```
C#
public class AddSecurityInitializerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the security initializer to set models of assets.
        AddSecurityInitializer(CustomSecurityInitializer);
    }

    private void CustomSecurityInitializer(Security security)
    {
        // Overwrite some of the reality models
        security.SetShortableProvider(new LocalDiskShortableProvider("axos"));
    }
}
```

To view all the pre-built shortable providers, see Supported Providers .

Default Behavior

The brokerage model of your algorithm automatically sets the shortable provider for each security. The default brokerage model is the `DefaultBrokerageModel` , which uses the NullShortableProvider .

Provider Structure

Shortable providers must implement the `IShortableProvider` interface. Shortable providers that implement the `IShortableProvider` interface must implement the `FeeRate` , `RebateRate` , and `ShortableQuantity` methods. These methods receives a Symbol and the local time of the algorithm. The `FeeRate` method returns the borrow fee rate. The `RebateRate` method returns the borrow rebate rate. The `ShortableQuantity` returns the shortable quantity.

```C#
public class CustomShortableProviderExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        var security = AddEquity("SPY");
        // Provide the selected security with your broker's shortable information
        security.SetShortableProvider(new MyShortableProvider());
    }
}

// Define the custom shortable provider
class MyShortableProvider : IShortableProvider
{
    public decimal FeeRate(Symbol symbol, DateTime localTime)
    {
        return 0.0025m;
    }

    public decimal RebateRate(Symbol symbol, DateTime localTime)
    {
        return 0.0507m;
    }

    public long? ShortableQuantity(Symbol symbol, DateTime localTime)
    {
        return 10000;
    }
}
```

Order Failures

If you place an order to short more shares than are available to borrow, LEAN rejects the order and applies the following tag: "Order exceeds maximum shortable quantity for Symbol ticker (requested short: quantity)".

Borrowing Costs

In live trading, your brokerage charges an interest fee when you borrow shares to short a security. In backtesting, use the ShortMarginInterestRateModel .

To get valid borrowing rates, use the InteractiveBrokersShortableProvider .

```C#
public override void Initialize()
{
    var security = AddEquity("SPY");
    // For US Equity, use the ShortMarginInterestRateModel with IB data
    security.SetShortableProvider(new InteractiveBrokersShortableProvider());
    security.MarginInterestRateModel = new ShortMarginInterestRateModel();
}
```

Examples

The following examples demonstrate common practices for using a shortable provider.

Example 1: Fixed Rates

The following algorithm defines a custom short availability model with fixed values for the fee rate, rebate rate, and shortable quantity.

```C#
public class CustomShortableProviderAlgorithm : QCAlgorithm
{
    private Equity _security;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        _security = AddEquity("QQQ", Resolution.Daily);
        _security.SetShortableProvider(new MyShortableProvider());
    }

    public override void OnData(Slice data)
    {
        if (Portfolio.Invested) return;
        var shortableQuantity = _security.ShortableProvider.ShortableQuantity(_security.Symbol, Time);
        if (shortableQuantity == null) return;
        var feeRate = _security.ShortableProvider.FeeRate(_security.Symbol, Time);
        var rebateRate = _security.ShortableProvider.RebateRate(_security.Symbol, Time);
        SetHoldings(_security.Symbol, -0.05m, tag: $"Borrow fee rate {feeRate}. Borrow rebate rate {rebateRate}");
    }
}

class MyShortableProvider : IShortableProvider
{
    public decimal FeeRate(Symbol symbol, DateTime localTime)
    {
        return 0.0025m;
    }

    public decimal RebateRate(Symbol symbol, DateTime localTime)
    {
        return 0.0507m;
    }

    public long? ShortableQuantity(Symbol symbol, DateTime localTime)
    {
        return 10000;
    }
}
```

Example 2: Sizing By Shortable Quantity

The following algorithm trades arbitrage of class A and class C price divergence of Google stocks using cointegration analysis. However, the position sizing may be limited by the available quantity to be shorted by the brokerage, breaking the paired trade. Thus, we need to make use of the `ShortableQuantity` method of the shortable provider to calculate the quantity.

```C#
using Accord.Statistics;
using MathNet.Numerics.LinearRegression;
using QuantConnect.Data.Shortable;

public class ShortableProviderAlgorithm : QCAlgorithm
{
    private Symbol _goog1, _goog2;
    // The threshold that the spread/residual of the cointegrated series triggers a trade.
    private decimal _thresold;
    // Store the coefficient and intercept of the cointegrated series for calculating the spread of a new data point.
    private decimal[] _coefficients = new[] { 0m, 0m };
    // Store the price series of each symbol for cointegration calculation.
    private Dictionary<Symbol, RollingWindow<decimal>> _windows = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Subscribe to 2 classes of Google stocks to trade their price divergence.
        var goog1 = AddEquity("GOOGL", Resolution.Daily);
        var goog2 = AddEquity("GOOG", Resolution.Daily);
        _goog1 = goog1.Symbol;          // Class A
        _goog2 = goog2.Symbol;          // Class C
```

```
        // Set the shortable providers to simulate the available shares to trade realistically.
        goog1.SetShortableProvider(new InteractiveBrokersShortableProvider());
        goog2.SetShortableProvider(new InteractiveBrokersShortableProvider());

        // Use rolling windows to save the price data for cointegration analysis.
        _windows[_goog1] = new(252);
        _windows[_goog2] = new(252);
        // Warm up the rolling windows.
        var history = History<TradeBar>(new[] { _goog1, _goog2 }, 252, Resolution.Daily);
        foreach (var bar in history)
        {
            _windows[_goog1].Add(bar[_goog1].Close);
            _windows[_goog2].Add(bar[_goog2].Close);
        }

        // Adjust the cointegration factor between the 2 classes' monthly price series.
        Schedule.On(
            DateRules.MonthStart(),
            TimeRules.At(0, 1),
            CalculateCointegration
        );

        CalculateCointegration();
    }

    public override void OnData(Slice slice)
    {
        if (slice.Bars.TryGetValue(_goog1, out var bar1) && slice.Bars.TryGetValue(_goog2, out var bar2))
        {
            // Update the rolling windows for cointegration analysis.
            _windows[_goog1].Add(bar1.Close);
            _windows[_goog2].Add(bar2.Close);

            // Calculate the current cointegrated series spread.
            var residual = _coefficients[0] * bar2.Close + _coefficients[1] - bar1.Close;

            var quantity = CalculateOrderQuantity(_goog1, 0.2m);
            // If the residual is lower than the negative threshold, it means class A price is much higher than what
it should be compared to class C.
            // We sell class A and buy class C to bet on their price convergence.
            if (residual < -_thresold && !Portfolio[_goog1].IsShort)
            {
                // Since it will be limited by shortable quantity, we max{shortable, 20% of portfolio value}.
                var shortableQuantity = Securities[_goog1].ShortableProvider.ShortableQuantity(_goog1, bar1.EndTime);
                quantity = Math.Min(quantity, Convert.ToDecimal(shortableQuantity));
                MarketOrder(_goog1, -quantity);
                MarketOrder(_goog2, quantity * _coefficients[0]);
            }
            // If the residual is higher than the threshold, it means class A price is much lower than what it should
be compared to class C.
            // We buy class A and sell class C to bet on their price convergence.
            else if (residual > _thresold && !Portfolio[_goog1].IsLong)
            {
                // Since it will be limited by shortable quantity, we max{shortable, 20% of portfolio value}.
                var shortableQuantity = Securities[_goog2].ShortableProvider.ShortableQuantity(_goog2, bar2.EndTime);
                quantity = Math.Min(quantity, Convert.ToDecimal(shortableQuantity));
                MarketOrder(_goog1, quantity);
                MarketOrder(_goog2, -quantity * _coefficients[0]);
            }
            // Close positions of the price are converged.
            else if ((Portfolio[_goog1].IsShort && residual > 0m) || (Portfolio[_goog1].IsLong && residual < 0m))
            {
                Liquidate();
            }
        }
    }

    private void CalculateCointegration()
    {
        // Lag direction is unimportant; it is just a sign flip in the linear regression, so we don't need to flip
the window order.
        var y = _windows[_goog1].Select(x => (double)x).ToArray();
        var x = _windows[_goog2].Select(x => (double)x).ToArray();

        // Perform Linear Regression on both price series to investigate their relationship.
        var regressionResult = SimpleRegression.Fit(x, y);
        var intercept = regressionResult.Item1;
        var slope = regressionResult.Item2;

        // Calculate the residuals series to check if it is stationary, meaning if the 2 price series move together.
        var residuals = new double[x.Length];
```

```
        for (int i = 0; i < x.Length; i++)
        {
            residuals[i] = y[i] - (intercept + slope * x[i]);
        }

        // Check if the residuals are stationary using the augmented Dickey-Fuller test.
        if (ADFTest(residuals))
        {
            // If cointegrated, update the positional sizing ratio and the spread threshold of the trade trigger.
            _coefficients = new[] { Convert.ToDecimal(slope), Convert.ToDecimal(intercept) };
            _thresold = 2m * Convert.ToDecimal(Measures.StandardDeviation(residuals));
        }
        else
        {
            // If not cointegrated, liquidate and set the size to zeros for no positions.
            Liquidate();
            _coefficients = new[] { 0m, 0m };
            _thresold = 100000000m;             // An arbitrarily large number that the class A price will never
reach.
        }
    }

    private static bool ADFTest(double[] series)
    {
        var n = series.Length;
        var lagged = new double[n - 1];
        var differences = new double[n - 1];

        // Fit linear regression for the residual series on unit root: ΔY_t = α + βY_{t-1} + ε_t.
        for (int i = 1; i < n; i++)
        {
            lagged[i - 1] = series[i - 1];
            differences[i - 1] = series[i] - series[i - 1];
        }

        var regressionResult = SimpleRegression.Fit(lagged, differences);
        var alpha = regressionResult.Item1;  // Intercept
        var beta = regressionResult.Item2;   // Coefficient of lagged term

        // Calculate the ADF statistic and check if the null hypothesis is rejected.
        var adfStatistic = beta / Measures.StandardError(differences);

        // Reject the null hypothesis of a unit root is present if test statistic <= -3.45 (approximate α=0.05 for
n=250)
        // This means no unit root exists for the difference series, and the residuals are stationary.
        return adfStatistic <= -3.45d;
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
InteractiveBrokersShortableProviderAlgorithm.cs C# CustomShortableProviderRegressionAlgorithm.cs C#
AllShortableSymbolsCoarseSelectionRegressionAlgorithm.cs C#

Short Availability

Supported Providers

Introduction

This page describes all of the pre-built shortable providers in LEAN. If none of these models perform exactly how you want, create a custom shortable provider .

Null Provider

The `NullShortableProvider` doesn't provide shortable data, so it allows you to infinitely short assets. It's the shortable provider of the `DefaultBrokerageModel` .

```
C#
security.SetShortableProvider(new NullShortableProvider());
```

For more information about this model, see the class reference and implementation .

Local Disk Provider

The `LocalDiskShortableProvider` sources short availability data from the local disk of a specific brokerage.

```
C#
security.SetShortableProvider(new LocalDiskShortableProvider("axos"));
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`brokerage`	`string`	Brokerage to read data	

For more information about this model, see the class reference and implementation .

Interactive Brokers Provider

The `InteractiveBrokersShortableProvider` sources short availability data from Interactive Brokers.

```
C#
security.SetShortableProvider(new InteractiveBrokersShortableProvider());
```

For more information about this model, see the class reference and implementation .

Scheduled Events

Introduction

Scheduled Events let you trigger code to run at specific times of day, regardless of your algorithm's data subscriptions. It's easier and more reliable to execute time-based events with Scheduled Events than checking the current algorithm time in the `OnData` event handler.

Create Scheduled Events

To create a Scheduled Event, call the `Schedule.On` method. The method expects a `DateRules` object, a `TimeRules` object, and a function to execute. The following examples demonstrate some common Scheduled Events.

Schedule Events Before Market Open

You may want to train a model or fetch historical data before the market opens. The following example demonstrates how to set a Scheduled Event for 10 minutes before the market opens.

```
C#
public override void Initialize()
{
    // Call the BeforeMarketOpen method 10 minutes before open every day SPY is trading.
    Schedule.On(
        DateRules.EveryDay("SPY"),
        TimeRules.BeforeMarketOpen("SPY", 10),
        BeforeMarketOpen
    );
}
```

Schedule Events on the Last Trading Day of the Week

You may want to rebalance your portfolio on the last trading day of each week and factor in market holidays. The following example demonstrates how to set a Scheduled Event for the last trading day of each week 30 minutes before the market closes.

```
C#
public override void Initialize()
{
    // Call the Rebalance method 30 minutes before the end of the SPY trading week.
    Schedule.On(
        DateRules.WeekEnd("SPY"),
        TimeRules.BeforeMarketClose("SPY", 30),
        Rebalance
    );
}
```

Schedule Events on Regular Intervals Throughout the Trading Day

You may want to perform some action on a regular interval through each trading day. The following example demonstrates how to set a Scheduled Event for every 30 minutes through the trading day for SPY.

```
C#
public override void Initialize()
{
    // Every day SPY is trading, call the 'SomeAction' method every 30 minutes.
    Schedule.On(
        DateRules.EveryDay("SPY"),
        TimeRules.Every(TimeSpan.FromMinutes(30)),
        SomeAction
    );
}
```

Date Rules

The following table describes the supported `DateRules` :

Member	Description
`DateRules.SetDefaultTimeZone(DateTimeZone timeZone);`	Sets the time zone for the `DateRules` object used in all methods in this table. The default time zone is the algorithm time zone .
`DateRules.On(int year, int month, int day)`	Trigger an event on a specific date.
`DateRules.On(params DateTime[] dates)`	Trigger an event on specific dates.
`DateRules.EveryDay()`	Trigger an event all seven days of the week.
`DateRules.EveryDay(Symbol symbol, bool extendedMarketHours = false)`	Trigger an event every day a specific symbol is trading.
`DateRules.Every(params DayOfWeek[] days)`	Trigger an event on specific days throughout the week. To view the `DayOfWeek` enum members, see DayOfWeek Enum in the .NET documentation.
`DateRules.MonthStart(int daysOffset = 0)`	Trigger an event on the first day of each month plus an offset.
`DateRules.MonthStart(Symbol symbol, int daysOffset = 0)`	Trigger an event on the first tradable date of each month for a specific symbol plus an offset.
`DateRules.MonthEnd(int daysOffset = 0)`	Trigger an event on the last day of each month minus an offset.
`DateRules.MonthEnd(Symbol symbol, int daysOffset = 0)`	Trigger an event on the last tradable date of each month for a specific symbol minus an offset.
`DateRules.WeekStart(int daysOffset = 0)`	Trigger an event on the first day of each week plus an offset.
`DateRules.WeekStart(Symbol symbol, int daysOffset = 0)`	Trigger an event on the first tradable date of each week for a specific symbol plus an offset.
`DateRules.WeekEnd(int daysOffset = 0)`	Trigger an event on the last day of each week minus an offset.
`DateRules.WeekEnd(Symbol symbol, int daysOffset = 0)`	Trigger an event on the last tradable date of each week for a specific symbol minus an offset.
`DateRules.YearStart(int daysOffset = 0)`	Trigger an event on the first day of each year plus an offset.
`DateRules.YearStart(Symbol symbol, int daysOffset = 0)`	Trigger an event on the first tradable date of each year for a specific symbol plus an offset.
`DateRules.YearEnd(int daysOffset = 0)`	Trigger an event on the last day of each year minus an offset.
`DateRules.YearEnd(Symbol symbol, int daysOffset = 0)`	Trigger an event on the last tradable date of each year for a specific symbol minus an offset.
`DateRules.Today`	Trigger an event once today.
`DateRules.Tomorrow`	Trigger an event once tomorrow.

To define custom date rules, create a `FuncDateRule` object. The `FuncDateRule` constructor expects a `name` argument of type `string` and a `getDatesFunction` argument of type `Func<DateTime, DateTime, IEnumerable<DateTime>>` . The `getDatesFunction` function receives the start and end dates of the algorithm and returns a list of dates for the date rule. In live trading, the end date is 12/31/2025. The following example demonstrates how to define a date rule that represents the 10th day of each month:

```
C#

// Create a date rule that specifies the 10th day of each month.
var dateRule = new FuncDateRule(
    name: "10th_day_of_the_month",
    getDatesFunction: (start, end) => Enumerable.Range(start.Year, end.Year - start.Year + 1)
        .SelectMany(year => Enumerable.Range(1, 12).Select(month => new DateTime(year, month, 10)))
);
```

Some date rules require a Symbol. However, they accept a string representation of the Symbol as well. If your algorithm doesn't add the security , it assumes the string refers to a US Equity Symbol. For example, if you use the `DateRules.EveryDay("ES")` method, it assumes that 'ES' refers to a US Equity. To create a Symbol without adding a security to your algorithm, call the `QuantConnect.Symbol.Create` method.

```
C#

var symbol = QuantConnect.Symbol.Create("ES", SecurityType.Future, Market.CME);
```

Time Rules

The following table describes the supported `TimeRules` :

Member	Description
`TimeRules.SetDefaultTimeZone(DateTimeZone timeZone)`	Sets the time zone for the `TimeRules` object used in all methods in this table, except when a different time zone is given. The default time zone is the algorithm time zone .
`TimeRules.BeforeMarketOpen(Symbol symbol, double minutesBeforeOpen = 0, bool extendedMarketOpen = false)`	Trigger an event a few minutes before market open for a specific symbol (default is 0). This rule doesn't work for Crypto securities or custom data.
`TimeRules.AfterMarketOpen(Symbol symbol, double minutesAfterOpen = 0, bool extendedMarketOpen = false)`	Trigger an event a few minutes after market open for a specific symbol (default is 0). This rule doesn't work for Crypto securities or custom data.
`TimeRules.BeforeMarketClose(Symbol symbol, double minutesBeforeClose = 0, bool extendedMarketOpen = false)`	Trigger an event a few minutes before market close for a specific symbol (default is 0). This rule doesn't work for Crypto securities or custom data.
`TimeRules.AfterMarketClose(Symbol symbol, double minutesAfterClose = 0, bool extendedMarketOpen = false)`	Trigger an event a few minutes after market close for a specific symbol (default is 0). This rule doesn't work for Crypto securities or custom data.
`TimeRules.Every(TimeSpan interval)`	Trigger an event every period interval starting at midnight. This time rule triggers every period, regardless of whether or not the market is open.
`TimeRules.Now`	Trigger an event at the current time of day.
`TimeRules.Midnight`	Trigger an event at midnight.
`TimeRules.Noon`	Trigger an event at noon.
`TimeRules.At(int hour, int minute, int second = 0)`	Trigger an event at a specific time of day (e.g. 13:10).
`TimeRules.At(int hour, int minute, int second, DateTimeZone timeZone)`	Trigger an event at a specific time of day in the given time zone (e.g. 13:10 UTC).

To define custom time rules, create a `FuncTimeRule` object. The `FuncTimeRule` constructor expects a `name` argument of type `string` and a `createUtcEventTimesFunction` argument of type `Func<IEnumerable<DateTime>, IEnumerable<DateTime>>` . The function receives the list of dates from the date rule and then returns a list of `DateTime` that define the time rule.

```
C#
var timeRule = new FuncTimeRule(
    name: "CustomTimeRule",
    createUtcEventTimesFunction: dates => dates.Select(d => d.AddHours(10)));
```

Some time rules require a Symbol. However, they accept a string representation of the Symbol as well. If your algorithm doesn't add the security , it assumes the string refers to a US Equity Symbol. For example, if you use the `TimeRules.BeforeMarketOpen("ES", 15)` method, it assumes that 'ES' refers to a US Equity. To create a Symbol without adding a security to your algorithm, call the `QuantConnect.Symbol.Create` method.

```
C#
var symbol = QuantConnect.Symbol.Create("ES", SecurityType.Future, Market.CME);
```

Remove Scheduled Events

If you no longer need a Scheduled Event in your algorithm, remove it so your algorithm doesn't execute unnecessary functions. To remove a Scheduled Event, save a reference to the Scheduled Event when you create it and then call the `Remove` method to remove it.

```
C#
private ScheduledEvent _scheduledEvent;

public override void Initialize()
{
    // Remove a scheduled event by keeping a copy of the object on creation.
    _scheduledEvent = Schedule.On(
        DateRules.EveryDay("SPY"),
        TimeRules.AfterMarketOpen("SPY", 10),
        TenMinutesAfterOpen
    );
}

// Remove the scheduled event using the object reference in any method
Schedule.Remove(_scheduledEvent);
```

Common Errors

Common errors with Scheduled Events include stale fills and execution timeouts.

Stale Fills

A common error is to subscribe to daily resolution data and set a Scheduled Event intraday to place trades. If you trade intraday with daily data, you get stale fills.

Another common error is to set a Scheduled Event to trade immediately after the market open on illiquid securities. Illiquid securities can have no trades for the first few minutes after the market open. If you trade during this time, you get stale fills.

Execution Timeouts

If your Scheduled Event takes longer than 10 minutes to execute, your algorithm will timeout. To increase the amount of time that your Scheduled Event can run, replace your Scheduled Event with a training session .

Invoking the Callback Method

Make sure to not accidentally invote the event handler method by adding trailing braces `()` . Use `MyCallbackFunction` , not `MyCallbackFunction()` .

Execution Sequence

The algorithm manager calls events in the following order:

1. Scheduled Events
2. Consolidation event handlers
3. `OnData` event handler

This event flow is important to note. For instance, if your consolidation handlers or `OnData` event handler appends data to a `RollingWindow` and you use that `RollingWindow` in your Scheduled Event, when the Scheduled Event executes, the `RollingWindow` won't contain the most recent data.

Live Trading Considerations

In live trading, Scheduled Events execute in a parallel thread based on a real-time clock. If you set a Scheduled Event to fire at 10:00 AM, it executes at exactly 10:00 AM. In backtesting, Scheduled Events are part of the main algorithm manager loop, so they may not execute exactly when you set them. For example, if your algorithm subscribes to minute resolution US Equity data

with regular trading hours and you set a Scheduled Event to occur at 2:00 AM, your Scheduled Event will execute at 9:31 AM when the next bar is fed into your algorithm.

The difference between live trading and backtesting is important to note because it can affect your algorithm's behavior. There are two common scenarios to consider.

Execution Timing and Backtest Timeouts

Take the following scenario:

- You set Scheduled Events for 2:00 AM, 3:00 AM, and 4:00 AM each day.
- Each Scheduled Event takes eight minutes to execute.
- Your algorithm only subscribes to US Equity securities without extended market hours (9:30 AM - 4:00 PM).

In this scenario, the Scheduled Events each fire at the correct time and execute without error in live trading. In backtesting, all of the Scheduled Events execute at 9:31 AM when your algorithm receives the first bar of the trading day. Since all the Scheduled Events take eight minutes to execute, the algorithm tries to execute all the Scheduled Events but reaches the 10-minute timeout and the backtest stops execution.

Live Data Delays

In backtests, your algorithm receives data at perfect timing. If you request minute resolution data, your algorithm receives the bars at the top of each minute. In live trading, bars have a slight delay, so you may receive them milliseconds after the top of each minute. Take the following scenario:

- You subscribe to minute resolution data
- You set a Scheduled Event for 10:00 AM
- The Scheduled Event checks the current asset price

In live trading, the Scheduled Event executes at exactly 10:00 AM but your algorithm may receive the 9:59-10:00 AM bar at 10:00:00.01 AM. Therefore, when you check the price in the Scheduled Event, the price from the 9:58-9:59 AM bar is the latest price. In backtesting, the Scheduled Event gets the price from the 9:59-10:00 AM bar since your algorithm receives the bar at perfect timing.

Examples

C#

```
public override void Initialize()
{
    var symbol = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);

    // Schedule an event to fire at a specific date/time using inline callback function.
    Schedule.On(DateRules.On(2013, 10, 7),
                TimeRules.At(13, 0),
                () => Log($"SpecificTime: Fired at : {Time}"));

    // Schedule an event to fire every trading day for a security
    // The time rule here tells it to fire at 13:00:00 UTC
    Schedule.On(DateRules.EveryDay("SPY"),
                TimeRules.At(13, 0, 0, TimeZones.Utc),
                () => Log($"EveryDay.SPY SpecificTime: Fired at: {Time}"));

    // Schedule an event to fire every trading day for a security
    // The time rule here tells it to fire 10 minutes before SPY's market open
    Schedule.On(DateRules.EveryDay("SPY"),
                TimeRules.BeforeMarketOpen("SPY", 10),
                () => Log($"EveryDay.SPY 10 min before open: Fired at: {Time}"));

    // Schedule an event to fire every trading day for a security
    // The time rule here tells it to fire 10 minutes after SPY's market open
    Schedule.On(DateRules.EveryDay("SPY"),
                TimeRules.AfterMarketOpen("SPY", 10),
                () => Log($"EveryDay.SPY 10 min after open: Fired at: {Time}"));
```

```
                () => Log($"EveryDay.SPY 10 min after open: Fired at: {Time}"));

    // Schedule an event to fire every trading day for a security
    // The time rule here tells it to fire 10 minutes before SPY's market close
    Schedule.On(DateRules.EveryDay("SPY"),
                TimeRules.BeforeMarketClose("SPY", 10),
                () => Log($"EveryDay.SPY 10 min before close: Fired at: {Time}"));

    // Schedule an event to fire every trading day for a security
    // The time rule here tells it to fire 10 minutes after SPY's market close
    Schedule.On(DateRules.EveryDay("SPY"),
                TimeRules.AfterMarketClose("SPY", 10),
                () => Log($"EveryDay.SPY 10 min after close: Fired at: {Time}"));

    // Schedule an event to fire on certain days of the week
    Schedule.On(DateRules.Every(DayOfWeek.Monday, DayOfWeek.Friday),
                TimeRules.At(12, 0),
                () => Log($"Mon/Fri at 12pm: Fired at: {Time}"));

    // Schedule an event to fire once today at when this method is called (now)
    Schedule.On(DateRules.Today,
                TimeRules.Now,
                () => Log($"Now: Fired at: {Time}"));

    // Schedule an event to fire once tomorrow at midnight
    Schedule.On(DateRules.Tomorrow,
                TimeRules.Midnight,
                () => Log($"Tomorrow at midnight: Fired at: {Time}"));

    // Schedule an event to fire once today at noon
    Schedule.On(DateRules.Today,
                TimeRules.Noon,
                () => Log($"Today at noon: Fired at: {Time}"));

    // The scheduling methods return the ScheduledEvent object which can be used
    // for other things here I set the event up to check the portfolio value every
    // 10 minutes, and liquidate if we have too many losses
    Schedule.On(DateRules.EveryDay(),
                TimeRules.Every(TimeSpan.FromMinutes(10)),
                LiquidateUnrealizedLosses);

    // Schedule an event to fire at the beginning of the month, the symbol is optional.
    // If specified, it will fire the first trading day for that symbol of the month.
    // Otherwise, it will fire on the first day of the month.
    Schedule.On(DateRules.MonthStart("SPY"),
                TimeRules.AfterMarketOpen("SPY"),
                RebalancingCode);

    // Schedule an event to fire at the end of the month, the symbol is optional.
    // If specified, it will fire the last trading day for that symbol of the month.
    // Otherwise, it will fire on the last day of the month.
    Schedule.On(DateRules.MonthEnd("SPY"),
                TimeRules.BeforeMarketClose("SPY"),
                RebalancingCode);

    // Schedule an event to fire at the beginning of the week, the symbol is optional.
    // If specified, it will fire the first trading day for that symbol of the week.
    // Otherwise, it will fire on the first day of the week.
    Schedule.On(DateRules.WeekStart("SPY"),
                TimeRules.AfterMarketOpen("SPY", 5),
                RebalancingCode);

    // Schedule an event to fire at the end of the week, the symbol is optional.
    // If specified, it will fire the last trading day for that symbol of the week.
    // Otherwise, it will fire on the first day of the week.
    Schedule.On(DateRules.WeekEnd("SPY"),
                TimeRules.BeforeMarketClose("SPY", 5),
                RebalancingCode);
    }

// The following methods are not defined in Initialize:
private void LiquidateUnrealizedLosses()
{
    // if we have over 1000 dollars in unrealized losses, liquidate
    if (Portfolio.TotalUnrealizedProfit < -1000)
    {
        Log($"Liquidated due to unrealized losses at: {Time}");
        Liquidate();
    }
}

private void RebalancingCode()
```

```
    private void RebalancingCode()
    {
        // Good spot for rebalancing code?
    }
```

Demonstration Algorithm
ScheduledEventsAlgorithm.cs C#

Indicators

Indicators

Supported Indicators

Indicators translate a stream of data points into a numerical value you can use to detect trading opportunities. LEAN provides more than 100 pre-built technical indicators and candlestick patterns you can use in your algorithms. You can use any of the following indicators. Click one to learn more.

Candlestick Patterns

Absolute Price Oscillator

Acceleration Bands

Accumulation Distribution

Accumulation Distribution Oscillator

Advance Decline Difference

Advance Decline Ratio

Advance Decline Volume Ratio

Alpha

Arms Index

Arnaud Legoux Moving Average

Aroon Oscillator

Augen Price Spike

Auto Regressive Integrated Moving Average

Average Directional Index

Average Directional Movement Index Rating

Average Range

Average True Range

Awesome Oscillator

Balance Of Power

Beta

Bollinger Bands

Chaikin Money Flow

Chaikin Oscillator

Chande Kroll Stop

Chande Momentum Oscillator

Choppiness Index

Commodity Channel Index

Connors Relative Strength Index

Coppock Curve

Correlation

Covariance

DeMarker Indicator

Delta

Derivative Oscillator

Detrended Price Oscillator

Donchian Channel

Double Exponential Moving Average

Ease Of Movement Value

Exponential Moving Average

Filtered Identity

Fisher Transform

Force Index

Fractal Adaptive Moving Average

Gamma

Heikin Ashi

Hilbert Transform

Hull Moving Average

Hurst Exponent

Ichimoku Kinko Hyo

Identity

Implied Volatility

Internal Bar Strength

Intraday Vwap

Kaufman Adaptive Moving Average

Kaufman Efficiency Ratio

Keltner Channels

Klinger Volume Oscillator

Know Sure Thing

Least Squares Moving Average

Linear Weighted Moving Average

Log Return

Mass Index

Maximum

McClellan Oscillator

McClellan Summation Index

McGinley Dynamic

Mean Absolute Deviation

Mesa Adaptive Moving Average

Mid Point

Mid Price

Minimum

Momentum

Momentum Percent

Momersion

Money Flow Index

Moving Average Convergence Divergence

New Highs New Lows

New Highs New Lows Volume

Normalized Average True Range

On Balance Volume

Parabolic Stop And Reverse

Parabolic Stop And Reverse Extended

Percentage Price Oscillator

Pivot Points High Low

Premier Stochastic Oscillator

Rate Of Change

Rate Of Change Percent

Rate Of Change Ratio

Regression Channel

Relative Daily Volume

Relative Moving Average

Relative Strength Index

Relative Vigor Index

Rho

Rogers Satchell Volatility

Schaff Trend Cycle

Sharpe Ratio

Simple Moving Average

Smoothed On Balance Volume

Sortino Ratio

Squeeze Momentum

Standard Deviation

Stochastic

Stochastic Relative Strength Index

Sum

Super Trend

Swiss Army Knife

T3 Moving Average

Target Downside Deviation

Theta

Time Profile

Time Series Forecast

Tom Demark Sequential

Triangular Moving Average

Triple Exponential Moving Average

Trix

True Range

True Strength Index

Ultimate Oscillator

Value At Risk

Variable Index Dynamic Average

Variance

Vega

Volume Profile

Volume Weighted Average Price Indicator

Volume Weighted Moving Average

Vortex

Wilder Accumulative Swing Index

Wilder Moving Average

Wilder Swing Index

Williams Percent R

Zero Lag Exponential Moving Average

Zig Zag

Supported Indicators

Candlestick Patterns

You can use any of the following candlestick patterns. Click one to learn more.

Abandoned Baby

Advance Block

Belt Hold

Breakaway

Closing Marubozu

Concealed Baby Swallow

Counterattack

Dark Cloud Cover

Doji

Doji Star

Dragonfly Doji

Engulfing

Evening Doji Star

Evening Star

Gap Side By Side White

Gravestone Doji

Hammer

Hanging Man

Harami

Harami Cross

High Wave Candle

Hikkake

Hikkake Modified

Homing Pigeon

Identical Three Crows

In Neck

Inverted Hammer

Kicking

Kicking By Length

Ladder Bottom

Long Legged Doji

Long Line Candle

Marubozu

Mat Hold

Matching Low

Morning Doji Star

Morning Star

On Neck

Piercing

Rickshaw Man

Rise Fall Three Methods

Separating Lines

Shooting Star

Short Line Candle

Spinning Top

Stalled Pattern

Stick Sandwich

Takuri

Tasuki Gap

Three Black Crows

Three Inside

Three Line Strike

Three Outside

Three Stars In South

Three White Soldiers

Thrusting

Tristar

Two Crows

Unique Three River

Up Down Gap Three Methods

Upside Gap Two Crows

Supported Indicators

Abandoned Baby

Introduction

Create a new Abandoned Baby candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using AbandonedBaby Indicator

To create an automatic indicator for `AbandonedBaby` , call the `AbandonedBaby` helper method from the `QCAlgorithm` class. The `AbandonedBaby` method creates a `AbandonedBaby` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class AbandonedBabyAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AbandonedBaby _abandonedBaby;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _abandonedBaby = CandlestickPatterns.AbandonedBaby(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_abandonedBaby.IsReady)
        {
            // The current value of _abandonedBaby is represented by itself (_abandonedBaby)
            // or _abandonedBaby.Current.Value
            Plot("AbandonedBaby", "abandonedBaby", _abandonedBaby);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `AbandonedBaby` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class AbandonedBabyAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AbandonedBaby _abandonedBaby;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _abandonedBaby = new AbandonedBaby();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _abandonedBaby.Update(bar);

        if (_abandonedBaby.IsReady)
        {
            // The current value of _abandonedBaby is represented by itself (_abandonedBaby)
            // or _abandonedBaby.Current.Value
            Plot("AbandonedBaby", "abandonedBaby", _abandonedBaby);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `AbandonedBaby` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class AbandonedBabyAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AbandonedBaby _abandonedBaby;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _abandonedBaby = CandlestickPatterns.AbandonedBaby(_symbol);

        var indicatorHistory = IndicatorHistory(_abandonedBaby, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_abandonedBaby, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_abandonedBaby, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Advance Block

Introduction

Create a new Advance Block candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using AdvanceBlock Indicator

To create an automatic indicator for `AdvanceBlock` , call the `AdvanceBlock` helper method from the `QCAlgorithm` class. The `AdvanceBlock` method creates a `AdvanceBlock` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class AdvanceBlockAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AdvanceBlock _advanceBlock;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _advanceBlock = CandlestickPatterns.AdvanceBlock(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_advanceBlock.IsReady)
        {
            // The current value of _advanceBlock is represented by itself (_advanceBlock)
            // or _advanceBlock.Current.Value
            Plot("AdvanceBlock", "advanceBlock", _advanceBlock);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `AdvanceBlock` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class AdvanceBlockAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AdvanceBlock _advanceBlock;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _advanceBlock = new AdvanceBlock();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _advanceBlock.Update(bar);

        if (_advanceBlock.IsReady)
        {
            // The current value of _advanceBlock is represented by itself (_advanceBlock)
            // or _advanceBlock.Current.Value
            Plot("AdvanceBlock", "advanceBlock", _advanceBlock);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `AdvanceBlock` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class AdvanceBlockAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AdvanceBlock _advanceBlock;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _advanceBlock = CandlestickPatterns.AdvanceBlock(_symbol);

        var indicatorHistory = IndicatorHistory(_advanceBlock, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_advanceBlock, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_advanceBlock, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Belt Hold

Introduction

Create a new Belt Hold candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using BeltHold Indicator

To create an automatic indicator for `BeltHold` , call the `BeltHold` helper method from the `QCAlgorithm` class. The `BeltHold` method creates a `BeltHold` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class BeltHoldAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private BeltHold _beltHold;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _beltHold = CandlestickPatterns.BeltHold(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_beltHold.IsReady)
        {
            // The current value of _beltHold is represented by itself (_beltHold)
            // or _beltHold.Current.Value
            Plot("BeltHold", "beltHold", _beltHold);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `BeltHold` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class BeltHoldAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private BeltHold _beltHold;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _beltHold = new BeltHold();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _beltHold.Update(bar);

        if (_beltHold.IsReady)
        {
            // The current value of _beltHold is represented by itself (_beltHold)
            // or _beltHold.Current.Value
            Plot("BeltHold", "beltHold", _beltHold);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `BeltHold` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class BeltHoldAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private BeltHold _beltHold;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _beltHold = CandlestickPatterns.BeltHold(_symbol);

        var indicatorHistory = IndicatorHistory(_beltHold, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_beltHold, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_beltHold, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Breakaway

Introduction

Create a new Breakaway candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Breakaway Indicator

To create an automatic indicator for `Breakaway` , call the `Breakaway` helper method from the `QCAlgorithm` class. The `Breakaway` method creates a `Breakaway` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class BreakawayAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Breakaway _breakaway;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _breakaway = CandlestickPatterns.Breakaway(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_breakaway.IsReady)
        {
            // The current value of _breakaway is represented by itself (_breakaway)
            // or _breakaway.Current.Value
            Plot("Breakaway", "breakaway", _breakaway);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `Breakaway` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class BreakawayAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Breakaway _breakaway;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _breakaway = new Breakaway();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _breakaway.Update(bar);

        if (_breakaway.IsReady)
        {
            // The current value of _breakaway is represented by itself (_breakaway)
            // or _breakaway.Current.Value
            Plot("Breakaway", "breakaway", _breakaway);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `Breakaway` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class BreakawayAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Breakaway _breakaway;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _breakaway = CandlestickPatterns.Breakaway(_symbol);

        var indicatorHistory = IndicatorHistory(_breakaway, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_breakaway, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_breakaway, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Closing Marubozu

Introduction

Create a new Closing Marubozu candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using ClosingMarubozu Indicator

To create an automatic indicator for `ClosingMarubozu` , call the `ClosingMarubozu` helper method from the `QCAlgorithm` class. The `ClosingMarubozu` method creates a `ClosingMarubozu` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class ClosingMarubozuAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ClosingMarubozu _closingMarubozu;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _closingMarubozu = CandlestickPatterns.ClosingMarubozu(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_closingMarubozu.IsReady)
        {
            // The current value of _closingMarubozu is represented by itself (_closingMarubozu)
            // or _closingMarubozu.Current.Value
            Plot("ClosingMarubozu", "closingMarubozu", _closingMarubozu);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `ClosingMarubozu` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class ClosingMarubozuAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ClosingMarubozu _closingMarubozu;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _closingMarubozu = new ClosingMarubozu();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _closingMarubozu.Update(bar);

        if (_closingMarubozu.IsReady)
        {
            // The current value of _closingMarubozu is represented by itself (_closingMarubozu)
            // or _closingMarubozu.Current.Value
            Plot("ClosingMarubozu", "closingMarubozu", _closingMarubozu);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `ClosingMarubozu` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class ClosingMarubozuAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ClosingMarubozu _closingMarubozu;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _closingMarubozu = CandlestickPatterns.ClosingMarubozu(_symbol);

        var indicatorHistory = IndicatorHistory(_closingMarubozu, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_closingMarubozu, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_closingMarubozu, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Concealed Baby Swallow

Introduction

Create a new Concealed Baby Swallow candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using ConcealedBabySwallow Indicator

To create an automatic indicator for `ConcealedBabySwallow` , call the `ConcealedBabySwallow` helper method from the `QCAlgorithm` class. The `ConcealedBabySwallow` method creates a `ConcealedBabySwallow` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class ConcealedBabySwallowAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ConcealedBabySwallow _concealedBabySwallow;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _concealedBabySwallow = CandlestickPatterns.ConcealedBabySwallow(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_concealedBabySwallow.IsReady)
        {
            // The current value of _concealedBabySwallow is represented by itself (_concealedBabySwallow)
            // or _concealedBabySwallow.Current.Value
            Plot("ConcealedBabySwallow", "concealedBabySwallow", _concealedBabySwallow);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `ConcealedBabySwallow` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ConcealedBabySwallowAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ConcealedBabySwallow _concealedBabySwallow;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _concealedBabySwallow = new ConcealedBabySwallow();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _concealedBabySwallow.Update(bar);

        if (_concealedBabySwallow.IsReady)
        {
            // The current value of _concealedBabySwallow is represented by itself (_concealedBabySwallow)
            // or _concealedBabySwallow.Current.Value
            Plot("ConcealedBabySwallow", "concealedBabySwallow", _concealedBabySwallow);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `ConcealedBabySwallow` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class ConcealedBabySwallowAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ConcealedBabySwallow _concealedBabySwallow;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _concealedBabySwallow = CandlestickPatterns.ConcealedBabySwallow(_symbol);

        var indicatorHistory = IndicatorHistory(_concealedBabySwallow, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_concealedBabySwallow, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_concealedBabySwallow, _symbol, new DateTime(2024, 7, 1),
new DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Counterattack

Introduction

Create a new Counterattack candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Counterattack Indicator

To create an automatic indicator for `Counterattack` , call the `Counterattack` helper method from the `QCAlgorithm` class. The `Counterattack` method creates a `Counterattack` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class CounterattackAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Counterattack _counterattack;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _counterattack = CandlestickPatterns.Counterattack(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_counterattack.IsReady)
        {
            // The current value of _counterattack is represented by itself (_counterattack)
            // or _counterattack.Current.Value
            Plot("Counterattack", "counterattack", _counterattack);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `Counterattack` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class CounterattackAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Counterattack _counterattack;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _counterattack = new Counterattack();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _counterattack.Update(bar);

        if (_counterattack.IsReady)
        {
            // The current value of _counterattack is represented by itself (_counterattack)
            // or _counterattack.Current.Value
            Plot("Counterattack", "counterattack", _counterattack);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `Counterattack` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class CounterattackAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Counterattack _counterattack;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _counterattack = CandlestickPatterns.Counterattack(_symbol);

        var indicatorHistory = IndicatorHistory(_counterattack, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_counterattack, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_counterattack, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Dark Cloud Cover

Introduction

Create a new Dark Cloud Cover candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using DarkCloudCover Indicator

To create an automatic indicator for `DarkCloudCover` , call the `DarkCloudCover` helper method from the `QCAlgorithm` class. The `DarkCloudCover` method creates a `DarkCloudCover` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class DarkCloudCoverAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DarkCloudCover _darkCloudCover;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _darkCloudCover = CandlestickPatterns.DarkCloudCover(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_darkCloudCover.IsReady)
        {
            // The current value of _darkCloudCover is represented by itself (_darkCloudCover)
            // or _darkCloudCover.Current.Value
            Plot("DarkCloudCover", "darkCloudCover", _darkCloudCover);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `DarkCloudCover` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class DarkCloudCoverAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DarkCloudCover _darkCloudCover;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _darkCloudCover = new DarkCloudCover();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _darkCloudCover.Update(bar);

        if (_darkCloudCover.IsReady)
        {
            // The current value of _darkCloudCover is represented by itself (_darkCloudCover)
            // or _darkCloudCover.Current.Value
            Plot("DarkCloudCover", "darkCloudCover", _darkCloudCover);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `DarkCloudCover` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class DarkCloudCoverAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DarkCloudCover _darkCloudCover;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _darkCloudCover = CandlestickPatterns.DarkCloudCover(_symbol);

        var indicatorHistory = IndicatorHistory(_darkCloudCover, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_darkCloudCover, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_darkCloudCover, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Doji

Introduction

Create a new Doji candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Doji Indicator

To create an automatic indicator for `Doji` , call the `Doji` helper method from the `QCAlgorithm` class. The `Doji` method creates a `Doji` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class DojiAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Doji _doji;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _doji = CandlestickPatterns.Doji(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_doji.IsReady)
        {
            // The current value of _doji is represented by itself (_doji)
            // or _doji.Current.Value
            Plot("Doji", "doji", _doji);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `Doji` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class DojiAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Doji _doji;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _doji = new Doji();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _doji.Update(bar);

        if (_doji.IsReady)
        {
            // The current value of _doji is represented by itself (_doji)
            // or _doji.Current.Value
            Plot("Doji", "doji", _doji);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `Doji` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class DojiAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Doji _doji;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _doji = CandlestickPatterns.Doji(_symbol);

        var indicatorHistory = IndicatorHistory(_doji, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_doji, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_doji, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Doji Star

Introduction

Create a new Doji Star candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using DojiStar Indicator

To create an automatic indicator for `DojiStar` , call the `DojiStar` helper method from the `QCAlgorithm` class. The `DojiStar` method creates a `DojiStar` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class DojiStarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DojiStar _dojiStar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _dojiStar = CandlestickPatterns.DojiStar(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_dojiStar.IsReady)
        {
            // The current value of _dojiStar is represented by itself (_dojiStar)
            // or _dojiStar.Current.Value
            Plot("DojiStar", "dojiStar", _dojiStar);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `DojiStar` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class DojiStarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DojiStar _dojiStar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _dojiStar = new DojiStar();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _dojiStar.Update(bar);

        if (_dojiStar.IsReady)
        {
            // The current value of _dojiStar is represented by itself (_dojiStar)
            // or _dojiStar.Current.Value
            Plot("DojiStar", "dojiStar", _dojiStar);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `DojiStar` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class DojiStarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DojiStar _dojiStar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _dojiStar = CandlestickPatterns.DojiStar(_symbol);

        var indicatorHistory = IndicatorHistory(_dojiStar, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_dojiStar, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_dojiStar, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Dragonfly Doji

Introduction

Create a new Dragonfly Doji candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using DragonflyDoji Indicator

To create an automatic indicator for `DragonflyDoji` , call the `DragonflyDoji` helper method from the `QCAlgorithm` class. The `DragonflyDoji` method creates a `DragonflyDoji` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class DragonflyDojiAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DragonflyDoji _dragonflyDoji;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _dragonflyDoji = CandlestickPatterns.DragonflyDoji(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_dragonflyDoji.IsReady)
        {
            // The current value of _dragonflyDoji is represented by itself (_dragonflyDoji)
            // or _dragonflyDoji.Current.Value
            Plot("DragonflyDoji", "dragonflyDoji", _dragonflyDoji);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `DragonflyDoji` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class DragonflyDojiAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DragonflyDoji _dragonflyDoji;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _dragonflyDoji = new DragonflyDoji();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _dragonflyDoji.Update(bar);

        if (_dragonflyDoji.IsReady)
        {
            // The current value of _dragonflyDoji is represented by itself (_dragonflyDoji)
            // or _dragonflyDoji.Current.Value
            Plot("DragonflyDoji", "dragonflyDoji", _dragonflyDoji);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `DragonflyDoji` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class DragonflyDojiAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DragonflyDoji _dragonflyDoji;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _dragonflyDoji = CandlestickPatterns.DragonflyDoji(_symbol);

        var indicatorHistory = IndicatorHistory(_dragonflyDoji, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_dragonflyDoji, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_dragonflyDoji, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Engulfing

Introduction

Create a new Engulfing candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Engulfing Indicator

To create an automatic indicator for `Engulfing` , call the `Engulfing` helper method from the `QCAlgorithm` class. The `Engulfing` method creates a `Engulfing` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class EngulfingAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Engulfing _engulfing;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _engulfing = CandlestickPatterns.Engulfing(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_engulfing.IsReady)
        {
            // The current value of _engulfing is represented by itself (_engulfing)
            // or _engulfing.Current.Value
            Plot("Engulfing", "engulfing", _engulfing);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `Engulfing` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class EngulfingAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Engulfing _engulfing;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _engulfing = new Engulfing();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _engulfing.Update(bar);

        if (_engulfing.IsReady)
        {
            // The current value of _engulfing is represented by itself (_engulfing)
            // or _engulfing.Current.Value
            Plot("Engulfing", "engulfing", _engulfing);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `Engulfing` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class EngulfingAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Engulfing _engulfing;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _engulfing = CandlestickPatterns.Engulfing(_symbol);

        var indicatorHistory = IndicatorHistory(_engulfing, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_engulfing, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_engulfing, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Evening Doji Star

Introduction

Create a new Evening Doji Star candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using EveningDojiStar Indicator

To create an automatic indicator for `EveningDojiStar` , call the `EveningDojiStar` helper method from the `QCAlgorithm` class. The `EveningDojiStar` method creates a `EveningDojiStar` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class EveningDojiStarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private EveningDojiStar _eveningDojiStar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _eveningDojiStar = CandlestickPatterns.EveningDojiStar(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_eveningDojiStar.IsReady)
        {
            // The current value of _eveningDojiStar is represented by itself (_eveningDojiStar)
            // or _eveningDojiStar.Current.Value
            Plot("EveningDojiStar", "eveningDojiStar", _eveningDojiStar);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `EveningDojiStar` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class EveningDojiStarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private EveningDojiStar _eveningDojiStar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _eveningDojiStar = new EveningDojiStar();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _eveningDojiStar.Update(bar);

        if (_eveningDojiStar.IsReady)
        {
            // The current value of _eveningDojiStar is represented by itself (_eveningDojiStar)
            // or _eveningDojiStar.Current.Value
            Plot("EveningDojiStar", "eveningDojiStar", _eveningDojiStar);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `EveningDojiStar` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class EveningDojiStarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private EveningDojiStar _eveningDojiStar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _eveningDojiStar = CandlestickPatterns.EveningDojiStar(_symbol);

        var indicatorHistory = IndicatorHistory(_eveningDojiStar, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_eveningDojiStar, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_eveningDojiStar, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Evening Star

Introduction

Create a new Evening Star candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using EveningStar Indicator

To create an automatic indicator for `EveningStar` , call the `EveningStar` helper method from the `QCAlgorithm` class. The `EveningStar` method creates a `EveningStar` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class EveningStarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private EveningStar _eveningStar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _eveningStar = CandlestickPatterns.EveningStar(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_eveningStar.IsReady)
        {
            // The current value of _eveningStar is represented by itself (_eveningStar)
            // or _eveningStar.Current.Value
            Plot("EveningStar", "eveningStar", _eveningStar);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `EveningStar` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class EveningStarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private EveningStar _eveningStar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _eveningStar = new EveningStar();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _eveningStar.Update(bar);

        if (_eveningStar.IsReady)
        {
            // The current value of _eveningStar is represented by itself (_eveningStar)
            // or _eveningStar.Current.Value
            Plot("EveningStar", "eveningStar", _eveningStar);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `EveningStar` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class EveningStarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private EveningStar _eveningStar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _eveningStar = CandlestickPatterns.EveningStar(_symbol);

        var indicatorHistory = IndicatorHistory(_eveningStar, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_eveningStar, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_eveningStar, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Gap Side By Side White

Introduction

Create a new Gap Side By Side White candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using GapSideBySideWhite Indicator

To create an automatic indicator for `GapSideBySideWhite` , call the `GapSideBySideWhite` helper method from the `QCAlgorithm` class. The `GapSideBySideWhite` method creates a `GapSideBySideWhite` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class GapSideBySideWhiteAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private GapSideBySideWhite _gapSideBySideWhite;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _gapSideBySideWhite = CandlestickPatterns.GapSideBySideWhite(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_gapSideBySideWhite.IsReady)
        {
            // The current value of _gapSideBySideWhite is represented by itself (_gapSideBySideWhite)
            // or _gapSideBySideWhite.Current.Value
            Plot("GapSideBySideWhite", "gapSideBySideWhite", _gapSideBySideWhite);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `GapSideBySideWhite` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class GapSideBySideWhiteAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private GapSideBySideWhite _gapSideBySideWhite;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _gapSideBySideWhite = new GapSideBySideWhite();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _gapSideBySideWhite.Update(bar);

        if (_gapSideBySideWhite.IsReady)
        {
            // The current value of _gapSideBySideWhite is represented by itself (_gapSideBySideWhite)
            // or _gapSideBySideWhite.Current.Value
            Plot("GapSideBySideWhite", "gapSideBySideWhite", _gapSideBySideWhite);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `GapSideBySideWhite` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class GapSideBySideWhiteAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private GapSideBySideWhite _gapSideBySideWhite;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _gapSideBySideWhite = CandlestickPatterns.GapSideBySideWhite(_symbol);

        var indicatorHistory = IndicatorHistory(_gapSideBySideWhite, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_gapSideBySideWhite, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_gapSideBySideWhite, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Gravestone Doji

Introduction

Create a new Gravestone Doji candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using GravestoneDoji Indicator

To create an automatic indicator for `GravestoneDoji` , call the `GravestoneDoji` helper method from the `QCAlgorithm` class. The `GravestoneDoji` method creates a `GravestoneDoji` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class GravestoneDojiAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private GravestoneDoji _gravestoneDoji;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _gravestoneDoji = CandlestickPatterns.GravestoneDoji(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_gravestoneDoji.IsReady)
        {
            // The current value of _gravestoneDoji is represented by itself (_gravestoneDoji)
            // or _gravestoneDoji.Current.Value
            Plot("GravestoneDoji", "gravestoneDoji", _gravestoneDoji);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `GravestoneDoji` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class GravestoneDojiAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private GravestoneDoji _gravestoneDoji;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _gravestoneDoji = new GravestoneDoji();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _gravestoneDoji.Update(bar);

        if (_gravestoneDoji.IsReady)
        {
            // The current value of _gravestoneDoji is represented by itself (_gravestoneDoji)
            // or _gravestoneDoji.Current.Value
            Plot("GravestoneDoji", "gravestoneDoji", _gravestoneDoji);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `GravestoneDoji` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class GravestoneDojiAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private GravestoneDoji _gravestoneDoji;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _gravestoneDoji = CandlestickPatterns.GravestoneDoji(_symbol);

        var indicatorHistory = IndicatorHistory(_gravestoneDoji, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_gravestoneDoji, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_gravestoneDoji, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Hammer

Introduction

Create a new Hammer candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Hammer Indicator

To create an automatic indicator for `Hammer` , call the `Hammer` helper method from the `QCAlgorithm` class. The `Hammer` method creates a `Hammer` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class HammerAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Hammer _hammer;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _hammer = CandlestickPatterns.Hammer(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_hammer.IsReady)
        {
            // The current value of _hammer is represented by itself (_hammer)
            // or _hammer.Current.Value
            Plot("Hammer", "hammer", _hammer);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `Hammer` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class HammerAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Hammer _hammer;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _hammer = new Hammer();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _hammer.Update(bar);

        if (_hammer.IsReady)
        {
            // The current value of _hammer is represented by itself (_hammer)
            // or _hammer.Current.Value
            Plot("Hammer", "hammer", _hammer);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `Hammer` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class HammerAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Hammer _hammer;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _hammer = CandlestickPatterns.Hammer(_symbol);

        var indicatorHistory = IndicatorHistory(_hammer, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_hammer, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_hammer, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Hanging Man

Introduction

Create a new Hanging Man candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using HangingMan Indicator

To create an automatic indicator for `HangingMan` , call the `HangingMan` helper method from the `QCAlgorithm` class. The `HangingMan` method creates a `HangingMan` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class HangingManAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HangingMan _hangingMan;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _hangingMan = CandlestickPatterns.HangingMan(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_hangingMan.IsReady)
        {
            // The current value of _hangingMan is represented by itself (_hangingMan)
            // or _hangingMan.Current.Value
            Plot("HangingMan", "hangingMan", _hangingMan);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `HangingMan` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class HangingManAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HangingMan _hangingMan;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _hangingMan = new HangingMan();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _hangingMan.Update(bar);

        if (_hangingMan.IsReady)
        {
            // The current value of _hangingMan is represented by itself (_hangingMan)
            // or _hangingMan.Current.Value
            Plot("HangingMan", "hangingMan", _hangingMan);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `HangingMan` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class HangingManAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HangingMan _hangingMan;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _hangingMan = CandlestickPatterns.HangingMan(_symbol);

        var indicatorHistory = IndicatorHistory(_hangingMan, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_hangingMan, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_hangingMan, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Harami

Introduction

Create a new Harami candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Harami Indicator

To create an automatic indicator for `Harami` , call the `Harami` helper method from the `QCAlgorithm` class. The `Harami` method creates a `Harami` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class HaramiAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Harami _harami;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _harami = CandlestickPatterns.Harami(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_harami.IsReady)
        {
            // The current value of _harami is represented by itself (_harami)
            // or _harami.Current.Value
            Plot("Harami", "harami", _harami);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `Harami` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class HaramiAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Harami _harami;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _harami = new Harami();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _harami.Update(bar);

        if (_harami.IsReady)
        {
            // The current value of _harami is represented by itself (_harami)
            // or _harami.Current.Value
            Plot("Harami", "harami", _harami);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `Harami` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class HaramiAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Harami _harami;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _harami = CandlestickPatterns.Harami(_symbol);

        var indicatorHistory = IndicatorHistory(_harami, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_harami, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_harami, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Harami Cross

Introduction

Create a new Harami Cross candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using HaramiCross Indicator

To create an automatic indicator for `HaramiCross` , call the `HaramiCross` helper method from the `QCAlgorithm` class. The `HaramiCross` method creates a `HaramiCross` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class HaramiCrossAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HaramiCross _haramiCross;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _haramiCross = CandlestickPatterns.HaramiCross(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_haramiCross.IsReady)
        {
            // The current value of _haramiCross is represented by itself (_haramiCross)
            // or _haramiCross.Current.Value
            Plot("HaramiCross", "haramiCross", _haramiCross);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `HaramiCross` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class HaramiCrossAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HaramiCross _haramiCross;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _haramiCross = new HaramiCross();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _haramiCross.Update(bar);

        if (_haramiCross.IsReady)
        {
            // The current value of _haramiCross is represented by itself (_haramiCross)
            // or _haramiCross.Current.Value
            Plot("HaramiCross", "haramiCross", _haramiCross);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `HaramiCross` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class HaramiCrossAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HaramiCross _haramiCross;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _haramiCross = CandlestickPatterns.HaramiCross(_symbol);

        var indicatorHistory = IndicatorHistory(_haramiCross, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_haramiCross, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_haramiCross, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

High Wave Candle

Introduction

Create a new High Wave Candle candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using HighWaveCandle Indicator

To create an automatic indicator for `HighWaveCandle` , call the `HighWaveCandle` helper method from the `QCAlgorithm` class. The `HighWaveCandle` method creates a `HighWaveCandle` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class HighWaveCandleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HighWaveCandle _highWaveCandle;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _highWaveCandle = CandlestickPatterns.HighWaveCandle(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_highWaveCandle.IsReady)
        {
            // The current value of _highWaveCandle is represented by itself (_highWaveCandle)
            // or _highWaveCandle.Current.Value
            Plot("HighWaveCandle", "highWaveCandle", _highWaveCandle);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `HighWaveCandle` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class HighWaveCandleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HighWaveCandle _highWaveCandle;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _highWaveCandle = new HighWaveCandle();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _highWaveCandle.Update(bar);

        if (_highWaveCandle.IsReady)
        {
            // The current value of _highWaveCandle is represented by itself (_highWaveCandle)
            // or _highWaveCandle.Current.Value
            Plot("HighWaveCandle", "highWaveCandle", _highWaveCandle);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `HighWaveCandle` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class HighWaveCandleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HighWaveCandle _highWaveCandle;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _highWaveCandle = CandlestickPatterns.HighWaveCandle(_symbol);

        var indicatorHistory = IndicatorHistory(_highWaveCandle, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_highWaveCandle, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_highWaveCandle, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Hikkake

Introduction

Create a new Hikkake candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Hikkake Indicator

To create an automatic indicator for `Hikkake` , call the `Hikkake` helper method from the `QCAlgorithm` class. The `Hikkake` method creates a `Hikkake` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class HikkakeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Hikkake _hikkake;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _hikkake = CandlestickPatterns.Hikkake(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_hikkake.IsReady)
        {
            // The current value of _hikkake is represented by itself (_hikkake)
            // or _hikkake.Current.Value
            Plot("Hikkake", "hikkake", _hikkake);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `Hikkake` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class HikkakeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Hikkake _hikkake;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _hikkake = new Hikkake();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _hikkake.Update(bar);

        if (_hikkake.IsReady)
        {
            // The current value of _hikkake is represented by itself (_hikkake)
            // or _hikkake.Current.Value
            Plot("Hikkake", "hikkake", _hikkake);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `Hikkake` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class HikkakeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Hikkake _hikkake;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _hikkake = CandlestickPatterns.Hikkake(_symbol);

        var indicatorHistory = IndicatorHistory(_hikkake, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_hikkake, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_hikkake, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Hikkake Modified

Introduction

Create a new Hikkake Modified candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using HikkakeModified Indicator

To create an automatic indicator for `HikkakeModified` , call the `HikkakeModified` helper method from the `QCAlgorithm` class. The `HikkakeModified` method creates a `HikkakeModified` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class HikkakeModifiedAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HikkakeModified _hikkakeModified;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _hikkakeModified = CandlestickPatterns.HikkakeModified(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_hikkakeModified.IsReady)
        {
            // The current value of _hikkakeModified is represented by itself (_hikkakeModified)
            // or _hikkakeModified.Current.Value
            Plot("HikkakeModified", "hikkakeModified", _hikkakeModified);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `HikkakeModified` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class HikkakeModifiedAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HikkakeModified _hikkakeModified;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _hikkakeModified = new HikkakeModified();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _hikkakeModified.Update(bar);

        if (_hikkakeModified.IsReady)
        {
            // The current value of _hikkakeModified is represented by itself (_hikkakeModified)
            // or _hikkakeModified.Current.Value
            Plot("HikkakeModified", "hikkakeModified", _hikkakeModified);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `HikkakeModified` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class HikkakeModifiedAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HikkakeModified _hikkakeModified;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _hikkakeModified = CandlestickPatterns.HikkakeModified(_symbol);

        var indicatorHistory = IndicatorHistory(_hikkakeModified, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_hikkakeModified, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_hikkakeModified, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Homing Pigeon

Introduction

Create a new Homing Pigeon candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using HomingPigeon Indicator

To create an automatic indicator for `HomingPigeon` , call the `HomingPigeon` helper method from the `QCAlgorithm` class. The `HomingPigeon` method creates a `HomingPigeon` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class HomingPigeonAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HomingPigeon _homingPigeon;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _homingPigeon = CandlestickPatterns.HomingPigeon(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_homingPigeon.IsReady)
        {
            // The current value of _homingPigeon is represented by itself (_homingPigeon)
            // or _homingPigeon.Current.Value
            Plot("HomingPigeon", "homingPigeon", _homingPigeon);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `HomingPigeon` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class HomingPigeonAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HomingPigeon _homingPigeon;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _homingPigeon = new HomingPigeon();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _homingPigeon.Update(bar);

        if (_homingPigeon.IsReady)
        {
            // The current value of _homingPigeon is represented by itself (_homingPigeon)
            // or _homingPigeon.Current.Value
            Plot("HomingPigeon", "homingPigeon", _homingPigeon);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `HomingPigeon` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class HomingPigeonAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HomingPigeon _homingPigeon;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _homingPigeon = CandlestickPatterns.HomingPigeon(_symbol);

        var indicatorHistory = IndicatorHistory(_homingPigeon, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_homingPigeon, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_homingPigeon, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Identical Three Crows

Introduction

Create a new Identical Three Crows candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using IdenticalThreeCrows Indicator

To create an automatic indicator for `IdenticalThreeCrows` , call the `IdenticalThreeCrows` helper method from the `QCAlgorithm` class. The `IdenticalThreeCrows` method creates a `IdenticalThreeCrows` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class IdenticalThreeCrowsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private IdenticalThreeCrows _identicalThreeCrows;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _identicalThreeCrows = CandlestickPatterns.IdenticalThreeCrows(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_identicalThreeCrows.IsReady)
        {
            // The current value of _identicalThreeCrows is represented by itself (_identicalThreeCrows)
            // or _identicalThreeCrows.Current.Value
            Plot("IdenticalThreeCrows", "identicalThreeCrows", _identicalThreeCrows);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `IdenticalThreeCrows` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class IdenticalThreeCrowsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private IdenticalThreeCrows _identicalThreeCrows;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _identicalThreeCrows = new IdenticalThreeCrows();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _identicalThreeCrows.Update(bar);

        if (_identicalThreeCrows.IsReady)
        {
            // The current value of _identicalThreeCrows is represented by itself (_identicalThreeCrows)
            // or _identicalThreeCrows.Current.Value
            Plot("IdenticalThreeCrows", "identicalThreeCrows", _identicalThreeCrows);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `IdenticalThreeCrows` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class IdenticalThreeCrowsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private IdenticalThreeCrows _identicalThreeCrows;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _identicalThreeCrows = CandlestickPatterns.IdenticalThreeCrows(_symbol);

        var indicatorHistory = IndicatorHistory(_identicalThreeCrows, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_identicalThreeCrows, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_identicalThreeCrows, _symbol, new DateTime(2024, 7, 1),
new DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

In Neck

Introduction

Create a new In Neck candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using InNeck Indicator

To create an automatic indicator for `InNeck` , call the `InNeck` helper method from the `QCAlgorithm` class. The `InNeck` method creates a `InNeck` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class InNeckAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private InNeck _inNeck;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _inNeck = CandlestickPatterns.InNeck(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_inNeck.IsReady)
        {
            // The current value of _inNeck is represented by itself (_inNeck)
            // or _inNeck.Current.Value
            Plot("InNeck", "inNeck", _inNeck);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `InNeck` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class InNeckAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private InNeck _inNeck;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _inNeck = new InNeck();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _inNeck.Update(bar);

        if (_inNeck.IsReady)
        {
            // The current value of _inNeck is represented by itself (_inNeck)
            // or _inNeck.Current.Value
            Plot("InNeck", "inNeck", _inNeck);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `InNeck` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class InNeckAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private InNeck _inNeck;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _inNeck = CandlestickPatterns.InNeck(_symbol);

        var indicatorHistory = IndicatorHistory(_inNeck, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_inNeck, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_inNeck, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Inverted Hammer

Introduction

Create a new Inverted Hammer candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using InvertedHammer Indicator

To create an automatic indicator for `InvertedHammer` , call the `InvertedHammer` helper method from the `QCAlgorithm` class. The `InvertedHammer` method creates a `InvertedHammer` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class InvertedHammerAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private InvertedHammer _invertedHammer;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _invertedHammer = CandlestickPatterns.InvertedHammer(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_invertedHammer.IsReady)
        {
            // The current value of _invertedHammer is represented by itself (_invertedHammer)
            // or _invertedHammer.Current.Value
            Plot("InvertedHammer", "invertedHammer", _invertedHammer);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `InvertedHammer` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class InvertedHammerAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private InvertedHammer _invertedHammer;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _invertedHammer = new InvertedHammer();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _invertedHammer.Update(bar);

        if (_invertedHammer.IsReady)
        {
            // The current value of _invertedHammer is represented by itself (_invertedHammer)
            // or _invertedHammer.Current.Value
            Plot("InvertedHammer", "invertedHammer", _invertedHammer);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `InvertedHammer` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class InvertedHammerAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private InvertedHammer _invertedHammer;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _invertedHammer = CandlestickPatterns.InvertedHammer(_symbol);

        var indicatorHistory = IndicatorHistory(_invertedHammer, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_invertedHammer, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_invertedHammer, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Kicking

Introduction

Create a new Kicking candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Kicking Indicator

To create an automatic indicator for `Kicking` , call the `Kicking` helper method from the `QCAlgorithm` class. The `Kicking` method creates a `Kicking` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class KickingAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Kicking _kicking;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _kicking = CandlestickPatterns.Kicking(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_kicking.IsReady)
        {
            // The current value of _kicking is represented by itself (_kicking)
            // or _kicking.Current.Value
            Plot("Kicking", "kicking", _kicking);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `Kicking` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class KickingAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Kicking _kicking;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _kicking = new Kicking();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _kicking.Update(bar);

        if (_kicking.IsReady)
        {
            // The current value of _kicking is represented by itself (_kicking)
            // or _kicking.Current.Value
            Plot("Kicking", "kicking", _kicking);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `Kicking` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class KickingAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Kicking _kicking;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _kicking = CandlestickPatterns.Kicking(_symbol);

        var indicatorHistory = IndicatorHistory(_kicking, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_kicking, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_kicking, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Kicking By Length

Introduction

Create a new Kicking By Length candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using KickingByLength Indicator

To create an automatic indicator for `KickingByLength` , call the `KickingByLength` helper method from the `QCAlgorithm` class. The `KickingByLength` method creates a `KickingByLength` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class KickingByLengthAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private KickingByLength _kickingByLength;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _kickingByLength = CandlestickPatterns.KickingByLength(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_kickingByLength.IsReady)
        {
            // The current value of _kickingByLength is represented by itself (_kickingByLength)
            // or _kickingByLength.Current.Value
            Plot("KickingByLength", "kickingByLength", _kickingByLength);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `KickingByLength` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class KickingByLengthAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private KickingByLength _kickingByLength;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _kickingByLength = new KickingByLength();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _kickingByLength.Update(bar);

        if (_kickingByLength.IsReady)
        {
            // The current value of _kickingByLength is represented by itself (_kickingByLength)
            // or _kickingByLength.Current.Value
            Plot("KickingByLength", "kickingByLength", _kickingByLength);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `KickingByLength` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class KickingByLengthAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private KickingByLength _kickingByLength;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _kickingByLength = CandlestickPatterns.KickingByLength(_symbol);

        var indicatorHistory = IndicatorHistory(_kickingByLength, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_kickingByLength, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_kickingByLength, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Ladder Bottom

Introduction

Create a new Ladder Bottom candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using LadderBottom Indicator

To create an automatic indicator for `LadderBottom` , call the `LadderBottom` helper method from the `QCAlgorithm` class. The `LadderBottom` method creates a `LadderBottom` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class LadderBottomAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private LadderBottom _ladderBottom;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ladderBottom = CandlestickPatterns.LadderBottom(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_ladderBottom.IsReady)
        {
            // The current value of _ladderBottom is represented by itself (_ladderBottom)
            // or _ladderBottom.Current.Value
            Plot("LadderBottom", "ladderBottom", _ladderBottom);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `LadderBottom` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class LadderBottomAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private LadderBottom _ladderBottom;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ladderBottom = new LadderBottom();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _ladderBottom.Update(bar);

        if (_ladderBottom.IsReady)
        {
            // The current value of _ladderBottom is represented by itself (_ladderBottom)
            // or _ladderBottom.Current.Value
            Plot("LadderBottom", "ladderBottom", _ladderBottom);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `LadderBottom` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class LadderBottomAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private LadderBottom _ladderBottom;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ladderBottom = CandlestickPatterns.LadderBottom(_symbol);

        var indicatorHistory = IndicatorHistory(_ladderBottom, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_ladderBottom, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_ladderBottom, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Long Legged Doji

Introduction

Create a new Long Legged Doji candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using LongLeggedDoji Indicator

To create an automatic indicator for `LongLeggedDoji` , call the `LongLeggedDoji` helper method from the `QCAlgorithm` class. The `LongLeggedDoji` method creates a `LongLeggedDoji` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class LongLeggedDojiAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private LongLeggedDoji _longLeggedDoji;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _longLeggedDoji = CandlestickPatterns.LongLeggedDoji(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_longLeggedDoji.IsReady)
        {
            // The current value of _longLeggedDoji is represented by itself (_longLeggedDoji)
            // or _longLeggedDoji.Current.Value
            Plot("LongLeggedDoji", "longLeggedDoji", _longLeggedDoji);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `LongLeggedDoji` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class LongLeggedDojiAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private LongLeggedDoji _longLeggedDoji;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _longLeggedDoji = new LongLeggedDoji();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _longLeggedDoji.Update(bar);

        if (_longLeggedDoji.IsReady)
        {
            // The current value of _longLeggedDoji is represented by itself (_longLeggedDoji)
            // or _longLeggedDoji.Current.Value
            Plot("LongLeggedDoji", "longLeggedDoji", _longLeggedDoji);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `LongLeggedDoji` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class LongLeggedDojiAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private LongLeggedDoji _longLeggedDoji;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _longLeggedDoji = CandlestickPatterns.LongLeggedDoji(_symbol);

        var indicatorHistory = IndicatorHistory(_longLeggedDoji, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_longLeggedDoji, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_longLeggedDoji, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Long Line Candle

Introduction

Create a new Long Line Candle candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using LongLineCandle Indicator

To create an automatic indicator for `LongLineCandle` , call the `LongLineCandle` helper method from the `QCAlgorithm` class. The `LongLineCandle` method creates a `LongLineCandle` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class LongLineCandleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private LongLineCandle _longLineCandle;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _longLineCandle = CandlestickPatterns.LongLineCandle(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_longLineCandle.IsReady)
        {
            // The current value of _longLineCandle is represented by itself (_longLineCandle)
            // or _longLineCandle.Current.Value
            Plot("LongLineCandle", "longLineCandle", _longLineCandle);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `LongLineCandle` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class LongLineCandleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private LongLineCandle _longLineCandle;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _longLineCandle = new LongLineCandle();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _longLineCandle.Update(bar);

        if (_longLineCandle.IsReady)
        {
            // The current value of _longLineCandle is represented by itself (_longLineCandle)
            // or _longLineCandle.Current.Value
            Plot("LongLineCandle", "longLineCandle", _longLineCandle);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `LongLineCandle` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class LongLineCandleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private LongLineCandle _longLineCandle;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _longLineCandle = CandlestickPatterns.LongLineCandle(_symbol);

        var indicatorHistory = IndicatorHistory(_longLineCandle, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_longLineCandle, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_longLineCandle, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Marubozu

Introduction

Create a new Marubozu candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Marubozu Indicator

To create an automatic indicator for `Marubozu` , call the `Marubozu` helper method from the `QCAlgorithm` class. The `Marubozu` method creates a `Marubozu` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class MarubozuAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Marubozu _marubozu;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _marubozu = CandlestickPatterns.Marubozu(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_marubozu.IsReady)
        {
            // The current value of _marubozu is represented by itself (_marubozu)
            // or _marubozu.Current.Value
            Plot("Marubozu", "marubozu", _marubozu);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `Marubozu` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class MarubozuAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Marubozu _marubozu;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _marubozu = new Marubozu();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _marubozu.Update(bar);

        if (_marubozu.IsReady)
        {
            // The current value of _marubozu is represented by itself (_marubozu)
            // or _marubozu.Current.Value
            Plot("Marubozu", "marubozu", _marubozu);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `Marubozu` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class MarubozuAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Marubozu _marubozu;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _marubozu = CandlestickPatterns.Marubozu(_symbol);

        var indicatorHistory = IndicatorHistory(_marubozu, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_marubozu, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_marubozu, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Mat Hold

Introduction

Create a new Mat Hold candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using MatHold Indicator

To create an automatic indicator for `MatHold` , call the `MatHold` helper method from the `QCAlgorithm` class. The `MatHold` method creates a `MatHold` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class MatHoldAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MatHold _matHold;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _matHold = CandlestickPatterns.MatHold(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_matHold.IsReady)
        {
            // The current value of _matHold is represented by itself (_matHold)
            // or _matHold.Current.Value
            Plot("MatHold", "matHold", _matHold);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `MatHold` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class MatHoldAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MatHold _matHold;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _matHold = new MatHold();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _matHold.Update(bar);

        if (_matHold.IsReady)
        {
            // The current value of _matHold is represented by itself (_matHold)
            // or _matHold.Current.Value
            Plot("MatHold", "matHold", _matHold);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `MatHold` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class MatHoldAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MatHold _matHold;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _matHold = CandlestickPatterns.MatHold(_symbol);

        var indicatorHistory = IndicatorHistory(_matHold, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_matHold, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_matHold, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Matching Low

Introduction

Create a new Matching Low candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using MatchingLow Indicator

To create an automatic indicator for `MatchingLow` , call the `MatchingLow` helper method from the `QCAlgorithm` class. The `MatchingLow` method creates a `MatchingLow` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class MatchingLowAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MatchingLow _matchingLow;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _matchingLow = CandlestickPatterns.MatchingLow(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_matchingLow.IsReady)
        {
            // The current value of _matchingLow is represented by itself (_matchingLow)
            // or _matchingLow.Current.Value
            Plot("MatchingLow", "matchingLow", _matchingLow);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `MatchingLow` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class MatchingLowAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MatchingLow _matchingLow;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _matchingLow = new MatchingLow();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _matchingLow.Update(bar);

        if (_matchingLow.IsReady)
        {
            // The current value of _matchingLow is represented by itself (_matchingLow)
            // or _matchingLow.Current.Value
            Plot("MatchingLow", "matchingLow", _matchingLow);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `MatchingLow` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class MatchingLowAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MatchingLow _matchingLow;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _matchingLow = CandlestickPatterns.MatchingLow(_symbol);

        var indicatorHistory = IndicatorHistory(_matchingLow, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_matchingLow, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_matchingLow, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Morning Doji Star

Introduction

Create a new Morning Doji Star candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using MorningDojiStar Indicator

To create an automatic indicator for `MorningDojiStar` , call the `MorningDojiStar` helper method from the `QCAlgorithm` class. The `MorningDojiStar` method creates a `MorningDojiStar` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class MorningDojiStarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MorningDojiStar _morningDojiStar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _morningDojiStar = CandlestickPatterns.MorningDojiStar(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_morningDojiStar.IsReady)
        {
            // The current value of _morningDojiStar is represented by itself (_morningDojiStar)
            // or _morningDojiStar.Current.Value
            Plot("MorningDojiStar", "morningDojiStar", _morningDojiStar);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `MorningDojiStar` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class MorningDojiStarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MorningDojiStar _morningDojiStar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _morningDojiStar = new MorningDojiStar();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _morningDojiStar.Update(bar);

        if (_morningDojiStar.IsReady)
        {
            // The current value of _morningDojiStar is represented by itself (_morningDojiStar)
            // or _morningDojiStar.Current.Value
            Plot("MorningDojiStar", "morningDojiStar", _morningDojiStar);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `MorningDojiStar` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class MorningDojiStarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MorningDojiStar _morningDojiStar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _morningDojiStar = CandlestickPatterns.MorningDojiStar(_symbol);

        var indicatorHistory = IndicatorHistory(_morningDojiStar, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_morningDojiStar, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_morningDojiStar, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Morning Star

Introduction

Create a new Morning Star candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using MorningStar Indicator

To create an automatic indicator for `MorningStar` , call the `MorningStar` helper method from the `QCAlgorithm` class. The `MorningStar` method creates a `MorningStar` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class MorningStarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MorningStar _morningStar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _morningStar = CandlestickPatterns.MorningStar(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_morningStar.IsReady)
        {
            // The current value of _morningStar is represented by itself (_morningStar)
            // or _morningStar.Current.Value
            Plot("MorningStar", "morningStar", _morningStar);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `MorningStar` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class MorningStarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MorningStar _morningStar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _morningStar = new MorningStar();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _morningStar.Update(bar);

        if (_morningStar.IsReady)
        {
            // The current value of _morningStar is represented by itself (_morningStar)
            // or _morningStar.Current.Value
            Plot("MorningStar", "morningStar", _morningStar);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `MorningStar` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class MorningStarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MorningStar _morningStar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _morningStar = CandlestickPatterns.MorningStar(_symbol);

        var indicatorHistory = IndicatorHistory(_morningStar, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_morningStar, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_morningStar, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

On Neck

Introduction

Create a new On Neck candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using OnNeck Indicator

To create an automatic indicator for `OnNeck` , call the `OnNeck` helper method from the `QCAlgorithm` class. The `OnNeck` method creates a `OnNeck` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class OnNeckAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private OnNeck _onNeck;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _onNeck = CandlestickPatterns.OnNeck(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_onNeck.IsReady)
        {
            // The current value of _onNeck is represented by itself (_onNeck)
            // or _onNeck.Current.Value
            Plot("OnNeck", "onNeck", _onNeck);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `OnNeck` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class OnNeckAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private OnNeck _onNeck;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _onNeck = new OnNeck();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _onNeck.Update(bar);

        if (_onNeck.IsReady)
        {
            // The current value of _onNeck is represented by itself (_onNeck)
            // or _onNeck.Current.Value
            Plot("OnNeck", "onNeck", _onNeck);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `OnNeck` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class OnNeckAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private OnNeck _onNeck;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _onNeck = CandlestickPatterns.OnNeck(_symbol);

        var indicatorHistory = IndicatorHistory(_onNeck, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_onNeck, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_onNeck, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Piercing

Introduction

Create a new Piercing candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Piercing Indicator

To create an automatic indicator for `Piercing` , call the `Piercing` helper method from the `QCAlgorithm` class. The `Piercing` method creates a `Piercing` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class PiercingAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Piercing _piercing;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _piercing = CandlestickPatterns.Piercing(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_piercing.IsReady)
        {
            // The current value of _piercing is represented by itself (_piercing)
            // or _piercing.Current.Value
            Plot("Piercing", "piercing", _piercing);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `Piercing` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class PiercingAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Piercing _piercing;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _piercing = new Piercing();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _piercing.Update(bar);

        if (_piercing.IsReady)
        {
            // The current value of _piercing is represented by itself (_piercing)
            // or _piercing.Current.Value
            Plot("Piercing", "piercing", _piercing);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `Piercing` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class PiercingAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Piercing _piercing;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _piercing = CandlestickPatterns.Piercing(_symbol);

        var indicatorHistory = IndicatorHistory(_piercing, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_piercing, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_piercing, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Rickshaw Man

Introduction

Create a new Rickshaw Man candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using RickshawMan Indicator

To create an automatic indicator for `RickshawMan` , call the `RickshawMan` helper method from the `QCAlgorithm` class. The `RickshawMan` method creates a `RickshawMan` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class RickshawManAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RickshawMan _rickshawMan;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rickshawMan = CandlestickPatterns.RickshawMan(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_rickshawMan.IsReady)
        {
            // The current value of _rickshawMan is represented by itself (_rickshawMan)
            // or _rickshawMan.Current.Value
            Plot("RickshawMan", "rickshawMan", _rickshawMan);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `RickshawMan` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class RickshawManAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RickshawMan _rickshawMan;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rickshawMan = new RickshawMan();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _rickshawMan.Update(bar);

        if (_rickshawMan.IsReady)
        {
            // The current value of _rickshawMan is represented by itself (_rickshawMan)
            // or _rickshawMan.Current.Value
            Plot("RickshawMan", "rickshawMan", _rickshawMan);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `RickshawMan` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class RickshawManAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RickshawMan _rickshawMan;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rickshawMan = CandlestickPatterns.RickshawMan(_symbol);

        var indicatorHistory = IndicatorHistory(_rickshawMan, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_rickshawMan, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_rickshawMan, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Rise Fall Three Methods

Introduction

Create a new Rise Fall Three Methods candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using RiseFallThreeMethods Indicator

To create an automatic indicator for `RiseFallThreeMethods` , call the `RiseFallThreeMethods` helper method from the `QCAlgorithm` class. The `RiseFallThreeMethods` method creates a `RiseFallThreeMethods` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class RiseFallThreeMethodsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RiseFallThreeMethods _riseFallThreeMethods;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _riseFallThreeMethods = CandlestickPatterns.RiseFallThreeMethods(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_riseFallThreeMethods.IsReady)
        {
            // The current value of _riseFallThreeMethods is represented by itself (_riseFallThreeMethods)
            // or _riseFallThreeMethods.Current.Value
            Plot("RiseFallThreeMethods", "riseFallThreeMethods", _riseFallThreeMethods);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `RiseFallThreeMethods` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class RiseFallThreeMethodsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RiseFallThreeMethods _riseFallThreeMethods;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _riseFallThreeMethods = new RiseFallThreeMethods();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _riseFallThreeMethods.Update(bar);

        if (_riseFallThreeMethods.IsReady)
        {
            // The current value of _riseFallThreeMethods is represented by itself (_riseFallThreeMethods)
            // or _riseFallThreeMethods.Current.Value
            Plot("RiseFallThreeMethods", "riseFallThreeMethods", _riseFallThreeMethods);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `RiseFallThreeMethods` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class RiseFallThreeMethodsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RiseFallThreeMethods _riseFallThreeMethods;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _riseFallThreeMethods = CandlestickPatterns.RiseFallThreeMethods(_symbol);

        var indicatorHistory = IndicatorHistory(_riseFallThreeMethods, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_riseFallThreeMethods, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_riseFallThreeMethods, _symbol, new DateTime(2024, 7, 1),
new DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Separating Lines

Introduction

Create a new Separating Lines candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using SeparatingLines Indicator

To create an automatic indicator for `SeparatingLines` , call the `SeparatingLines` helper method from the `QCAlgorithm` class. The `SeparatingLines` method creates a `SeparatingLines` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class SeparatingLinesAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SeparatingLines _separatingLines;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _separatingLines = CandlestickPatterns.SeparatingLines(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_separatingLines.IsReady)
        {
            // The current value of _separatingLines is represented by itself (_separatingLines)
            // or _separatingLines.Current.Value
            Plot("SeparatingLines", "separatingLines", _separatingLines);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `SeparatingLines` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class SeparatingLinesAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SeparatingLines _separatingLines;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _separatingLines = new SeparatingLines();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _separatingLines.Update(bar);

        if (_separatingLines.IsReady)
        {
            // The current value of _separatingLines is represented by itself (_separatingLines)
            // or _separatingLines.Current.Value
            Plot("SeparatingLines", "separatingLines", _separatingLines);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `SeparatingLines` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class SeparatingLinesAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SeparatingLines _separatingLines;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _separatingLines = CandlestickPatterns.SeparatingLines(_symbol);

        var indicatorHistory = IndicatorHistory(_separatingLines, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_separatingLines, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_separatingLines, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Shooting Star

Introduction

Create a new Shooting Star candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using ShootingStar Indicator

To create an automatic indicator for `ShootingStar` , call the `ShootingStar` helper method from the `QCAlgorithm` class. The `ShootingStar` method creates a `ShootingStar` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class ShootingStarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ShootingStar _shootingStar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _shootingStar = CandlestickPatterns.ShootingStar(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_shootingStar.IsReady)
        {
            // The current value of _shootingStar is represented by itself (_shootingStar)
            // or _shootingStar.Current.Value
            Plot("ShootingStar", "shootingStar", _shootingStar);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `ShootingStar` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ShootingStarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ShootingStar _shootingStar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _shootingStar = new ShootingStar();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _shootingStar.Update(bar);

        if (_shootingStar.IsReady)
        {
            // The current value of _shootingStar is represented by itself (_shootingStar)
            // or _shootingStar.Current.Value
            Plot("ShootingStar", "shootingStar", _shootingStar);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `ShootingStar` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class ShootingStarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ShootingStar _shootingStar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _shootingStar = CandlestickPatterns.ShootingStar(_symbol);

        var indicatorHistory = IndicatorHistory(_shootingStar, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_shootingStar, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_shootingStar, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Short Line Candle

Introduction

Create a new Short Line Candle candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using ShortLineCandle Indicator

To create an automatic indicator for `ShortLineCandle` , call the `ShortLineCandle` helper method from the `QCAlgorithm` class. The `ShortLineCandle` method creates a `ShortLineCandle` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class ShortLineCandleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ShortLineCandle _shortLineCandle;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _shortLineCandle = CandlestickPatterns.ShortLineCandle(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_shortLineCandle.IsReady)
        {
            // The current value of _shortLineCandle is represented by itself (_shortLineCandle)
            // or _shortLineCandle.Current.Value
            Plot("ShortLineCandle", "shortLineCandle", _shortLineCandle);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `ShortLineCandle` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ShortLineCandleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ShortLineCandle _shortLineCandle;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _shortLineCandle = new ShortLineCandle();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _shortLineCandle.Update(bar);

        if (_shortLineCandle.IsReady)
        {
            // The current value of _shortLineCandle is represented by itself (_shortLineCandle)
            // or _shortLineCandle.Current.Value
            Plot("ShortLineCandle", "shortLineCandle", _shortLineCandle);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `ShortLineCandle` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class ShortLineCandleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ShortLineCandle _shortLineCandle;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _shortLineCandle = CandlestickPatterns.ShortLineCandle(_symbol);

        var indicatorHistory = IndicatorHistory(_shortLineCandle, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_shortLineCandle, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_shortLineCandle, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Spinning Top

Introduction

Create a new Spinning Top candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using SpinningTop Indicator

To create an automatic indicator for `SpinningTop` , call the `SpinningTop` helper method from the `QCAlgorithm` class. The `SpinningTop` method creates a `SpinningTop` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class SpinningTopAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SpinningTop _spinningTop;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _spinningTop = CandlestickPatterns.SpinningTop(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_spinningTop.IsReady)
        {
            // The current value of _spinningTop is represented by itself (_spinningTop)
            // or _spinningTop.Current.Value
            Plot("SpinningTop", "spinningTop", _spinningTop);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `SpinningTop` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class SpinningTopAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SpinningTop _spinningTop;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _spinningTop = new SpinningTop();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _spinningTop.Update(bar);

        if (_spinningTop.IsReady)
        {
            // The current value of _spinningTop is represented by itself (_spinningTop)
            // or _spinningTop.Current.Value
            Plot("SpinningTop", "spinningTop", _spinningTop);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `SpinningTop` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class SpinningTopAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SpinningTop _spinningTop;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _spinningTop = CandlestickPatterns.SpinningTop(_symbol);

        var indicatorHistory = IndicatorHistory(_spinningTop, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_spinningTop, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_spinningTop, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Stalled Pattern

Introduction

Create a new Stalled Pattern candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using StalledPattern Indicator

To create an automatic indicator for `StalledPattern` , call the `StalledPattern` helper method from the `QCAlgorithm` class. The `StalledPattern` method creates a `StalledPattern` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class StalledPatternAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private StalledPattern _stalledPattern;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _stalledPattern = CandlestickPatterns.StalledPattern(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_stalledPattern.IsReady)
        {
            // The current value of _stalledPattern is represented by itself (_stalledPattern)
            // or _stalledPattern.Current.Value
            Plot("StalledPattern", "stalledPattern", _stalledPattern);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `StalledPattern` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class StalledPatternAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private StalledPattern _stalledPattern;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _stalledPattern = new StalledPattern();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _stalledPattern.Update(bar);

        if (_stalledPattern.IsReady)
        {
            // The current value of _stalledPattern is represented by itself (_stalledPattern)
            // or _stalledPattern.Current.Value
            Plot("StalledPattern", "stalledPattern", _stalledPattern);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `StalledPattern` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class StalledPatternAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private StalledPattern _stalledPattern;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _stalledPattern = CandlestickPatterns.StalledPattern(_symbol);

        var indicatorHistory = IndicatorHistory(_stalledPattern, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_stalledPattern, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_stalledPattern, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Stick Sandwich

Introduction

Create a new Stick Sandwich candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using StickSandwich Indicator

To create an automatic indicator for `StickSandwich` , call the `StickSandwich` helper method from the `QCAlgorithm` class. The `StickSandwich` method creates a `StickSandwich` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class StickSandwichAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private StickSandwich _stickSandwich;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _stickSandwich = CandlestickPatterns.StickSandwich(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_stickSandwich.IsReady)
        {
            // The current value of _stickSandwich is represented by itself (_stickSandwich)
            // or _stickSandwich.Current.Value
            Plot("StickSandwich", "stickSandwich", _stickSandwich);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `StickSandwich` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class StickSandwichAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private StickSandwich _stickSandwich;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _stickSandwich = new StickSandwich();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _stickSandwich.Update(bar);

        if (_stickSandwich.IsReady)
        {
            // The current value of _stickSandwich is represented by itself (_stickSandwich)
            // or _stickSandwich.Current.Value
            Plot("StickSandwich", "stickSandwich", _stickSandwich);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `StickSandwich` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class StickSandwichAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private StickSandwich _stickSandwich;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _stickSandwich = CandlestickPatterns.StickSandwich(_symbol);

        var indicatorHistory = IndicatorHistory(_stickSandwich, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_stickSandwich, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_stickSandwich, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Takuri

Introduction

Create a new Takuri candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Takuri Indicator

To create an automatic indicator for `Takuri` , call the `Takuri` helper method from the `QCAlgorithm` class. The `Takuri` method creates a `Takuri` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class TakuriAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Takuri _takuri;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _takuri = CandlestickPatterns.Takuri(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_takuri.IsReady)
        {
            // The current value of _takuri is represented by itself (_takuri)
            // or _takuri.Current.Value
            Plot("Takuri", "takuri", _takuri);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `Takuri` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class TakuriAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Takuri _takuri;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _takuri = new Takuri();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _takuri.Update(bar);

        if (_takuri.IsReady)
        {
            // The current value of _takuri is represented by itself (_takuri)
            // or _takuri.Current.Value
            Plot("Takuri", "takuri", _takuri);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `Takuri` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class TakuriAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Takuri _takuri;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _takuri = CandlestickPatterns.Takuri(_symbol);

        var indicatorHistory = IndicatorHistory(_takuri, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_takuri, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_takuri, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Tasuki Gap

Introduction

Create a new Tasuki Gap candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using TasukiGap Indicator

To create an automatic indicator for `TasukiGap` , call the `TasukiGap` helper method from the `QCAlgorithm` class. The `TasukiGap` method creates a `TasukiGap` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class TasukiGapAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TasukiGap _tasukiGap;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tasukiGap = CandlestickPatterns.TasukiGap(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_tasukiGap.IsReady)
        {
            // The current value of _tasukiGap is represented by itself (_tasukiGap)
            // or _tasukiGap.Current.Value
            Plot("TasukiGap", "tasukiGap", _tasukiGap);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `TasukiGap` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class TasukiGapAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TasukiGap _tasukiGap;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tasukiGap = new TasukiGap();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _tasukiGap.Update(bar);

        if (_tasukiGap.IsReady)
        {
            // The current value of _tasukiGap is represented by itself (_tasukiGap)
            // or _tasukiGap.Current.Value
            Plot("TasukiGap", "tasukiGap", _tasukiGap);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `TasukiGap` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class TasukiGapAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TasukiGap _tasukiGap;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tasukiGap = CandlestickPatterns.TasukiGap(_symbol);

        var indicatorHistory = IndicatorHistory(_tasukiGap, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_tasukiGap, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_tasukiGap, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Three Black Crows

Introduction

Create a new Three Black Crows candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using ThreeBlackCrows Indicator

To create an automatic indicator for `ThreeBlackCrows` , call the `ThreeBlackCrows` helper method from the `QCAlgorithm` class. The `ThreeBlackCrows` method creates a `ThreeBlackCrows` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class ThreeBlackCrowsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ThreeBlackCrows _threeBlackCrows;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _threeBlackCrows = CandlestickPatterns.ThreeBlackCrows(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_threeBlackCrows.IsReady)
        {
            // The current value of _threeBlackCrows is represented by itself (_threeBlackCrows)
            // or _threeBlackCrows.Current.Value
            Plot("ThreeBlackCrows", "threeBlackCrows", _threeBlackCrows);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `ThreeBlackCrows` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ThreeBlackCrowsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ThreeBlackCrows _threeBlackCrows;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _threeBlackCrows = new ThreeBlackCrows();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _threeBlackCrows.Update(bar);

        if (_threeBlackCrows.IsReady)
        {
            // The current value of _threeBlackCrows is represented by itself (_threeBlackCrows)
            // or _threeBlackCrows.Current.Value
            Plot("ThreeBlackCrows", "threeBlackCrows", _threeBlackCrows);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `ThreeBlackCrows` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class ThreeBlackCrowsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ThreeBlackCrows _threeBlackCrows;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _threeBlackCrows = CandlestickPatterns.ThreeBlackCrows(_symbol);

        var indicatorHistory = IndicatorHistory(_threeBlackCrows, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_threeBlackCrows, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_threeBlackCrows, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Three Inside

Introduction

Create a new Three Inside candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using ThreeInside Indicator

To create an automatic indicator for `ThreeInside` , call the `ThreeInside` helper method from the `QCAlgorithm` class. The `ThreeInside` method creates a `ThreeInside` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class ThreeInsideAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ThreeInside _threeInside;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _threeInside = CandlestickPatterns.ThreeInside(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_threeInside.IsReady)
        {
            // The current value of _threeInside is represented by itself (_threeInside)
            // or _threeInside.Current.Value
            Plot("ThreeInside", "threeInside", _threeInside);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `ThreeInside` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class ThreeInsideAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ThreeInside _threeInside;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _threeInside = new ThreeInside();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _threeInside.Update(bar);

        if (_threeInside.IsReady)
        {
            // The current value of _threeInside is represented by itself (_threeInside)
            // or _threeInside.Current.Value
            Plot("ThreeInside", "threeInside", _threeInside);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `ThreeInside` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class ThreeInsideAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ThreeInside _threeInside;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _threeInside = CandlestickPatterns.ThreeInside(_symbol);

        var indicatorHistory = IndicatorHistory(_threeInside, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_threeInside, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_threeInside, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Three Line Strike

Introduction

Create a new Three Line Strike candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using ThreeLineStrike Indicator

To create an automatic indicator for `ThreeLineStrike` , call the `ThreeLineStrike` helper method from the `QCAlgorithm` class. The `ThreeLineStrike` method creates a `ThreeLineStrike` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class ThreeLineStrikeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ThreeLineStrike _threeLineStrike;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _threeLineStrike = CandlestickPatterns.ThreeLineStrike(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_threeLineStrike.IsReady)
        {
            // The current value of _threeLineStrike is represented by itself (_threeLineStrike)
            // or _threeLineStrike.Current.Value
            Plot("ThreeLineStrike", "threeLineStrike", _threeLineStrike);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `ThreeLineStrike` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ThreeLineStrikeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ThreeLineStrike _threeLineStrike;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _threeLineStrike = new ThreeLineStrike();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _threeLineStrike.Update(bar);

        if (_threeLineStrike.IsReady)
        {
            // The current value of _threeLineStrike is represented by itself (_threeLineStrike)
            // or _threeLineStrike.Current.Value
            Plot("ThreeLineStrike", "threeLineStrike", _threeLineStrike);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `ThreeLineStrike` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class ThreeLineStrikeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ThreeLineStrike _threeLineStrike;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _threeLineStrike = CandlestickPatterns.ThreeLineStrike(_symbol);

        var indicatorHistory = IndicatorHistory(_threeLineStrike, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_threeLineStrike, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_threeLineStrike, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Three Outside

Introduction

Create a new Three Outside candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using ThreeOutside Indicator

To create an automatic indicator for `ThreeOutside` , call the `ThreeOutside` helper method from the `QCAlgorithm` class. The `ThreeOutside` method creates a `ThreeOutside` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class ThreeOutsideAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ThreeOutside _threeOutside;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _threeOutside = CandlestickPatterns.ThreeOutside(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_threeOutside.IsReady)
        {
            // The current value of _threeOutside is represented by itself (_threeOutside)
            // or _threeOutside.Current.Value
            Plot("ThreeOutside", "threeOutside", _threeOutside);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `ThreeOutside` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ThreeOutsideAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ThreeOutside _threeOutside;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _threeOutside = new ThreeOutside();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _threeOutside.Update(bar);

        if (_threeOutside.IsReady)
        {
            // The current value of _threeOutside is represented by itself (_threeOutside)
            // or _threeOutside.Current.Value
            Plot("ThreeOutside", "threeOutside", _threeOutside);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `ThreeOutside` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class ThreeOutsideAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ThreeOutside _threeOutside;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _threeOutside = CandlestickPatterns.ThreeOutside(_symbol);

        var indicatorHistory = IndicatorHistory(_threeOutside, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_threeOutside, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_threeOutside, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Three Stars In South

Introduction

Create a new Three Stars In South candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using ThreeStarsInSouth Indicator

To create an automatic indicator for `ThreeStarsInSouth` , call the `ThreeStarsInSouth` helper method from the `QCAlgorithm` class. The `ThreeStarsInSouth` method creates a `ThreeStarsInSouth` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class ThreeStarsInSouthAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ThreeStarsInSouth _threeStarsInSouth;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _threeStarsInSouth = CandlestickPatterns.ThreeStarsInSouth(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_threeStarsInSouth.IsReady)
        {
            // The current value of _threeStarsInSouth is represented by itself (_threeStarsInSouth)
            // or _threeStarsInSouth.Current.Value
            Plot("ThreeStarsInSouth", "threeStarsInSouth", _threeStarsInSouth);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `ThreeStarsInSouth` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ThreeStarsInSouthAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ThreeStarsInSouth _threeStarsInSouth;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _threeStarsInSouth = new ThreeStarsInSouth();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _threeStarsInSouth.Update(bar);

        if (_threeStarsInSouth.IsReady)
        {
            // The current value of _threeStarsInSouth is represented by itself (_threeStarsInSouth)
            // or _threeStarsInSouth.Current.Value
            Plot("ThreeStarsInSouth", "threeStarsInSouth", _threeStarsInSouth);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `ThreeStarsInSouth` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class ThreeStarsInSouthAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ThreeStarsInSouth _threeStarsInSouth;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _threeStarsInSouth = CandlestickPatterns.ThreeStarsInSouth(_symbol);

        var indicatorHistory = IndicatorHistory(_threeStarsInSouth, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_threeStarsInSouth, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_threeStarsInSouth, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Three White Soldiers

Introduction

Create a new Three White Soldiers candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using ThreeWhiteSoldiers Indicator

To create an automatic indicator for `ThreeWhiteSoldiers` , call the `ThreeWhiteSoldiers` helper method from the `QCAlgorithm` class. The `ThreeWhiteSoldiers` method creates a `ThreeWhiteSoldiers` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

C#

```
public class ThreeWhiteSoldiersAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ThreeWhiteSoldiers _threeWhiteSoldiers;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _threeWhiteSoldiers = CandlestickPatterns.ThreeWhiteSoldiers(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_threeWhiteSoldiers.IsReady)
        {
            // The current value of _threeWhiteSoldiers is represented by itself (_threeWhiteSoldiers)
            // or _threeWhiteSoldiers.Current.Value
            Plot("ThreeWhiteSoldiers", "threeWhiteSoldiers", _threeWhiteSoldiers);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `ThreeWhiteSoldiers` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ThreeWhiteSoldiersAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ThreeWhiteSoldiers _threeWhiteSoldiers;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _threeWhiteSoldiers = new ThreeWhiteSoldiers();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _threeWhiteSoldiers.Update(bar);

        if (_threeWhiteSoldiers.IsReady)
        {
            // The current value of _threeWhiteSoldiers is represented by itself (_threeWhiteSoldiers)
            // or _threeWhiteSoldiers.Current.Value
            Plot("ThreeWhiteSoldiers", "threeWhiteSoldiers", _threeWhiteSoldiers);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `ThreeWhiteSoldiers` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class ThreeWhiteSoldiersAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ThreeWhiteSoldiers _threeWhiteSoldiers;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _threeWhiteSoldiers = CandlestickPatterns.ThreeWhiteSoldiers(_symbol);

        var indicatorHistory = IndicatorHistory(_threeWhiteSoldiers, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_threeWhiteSoldiers, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_threeWhiteSoldiers, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Thrusting

Introduction

Create a new Thrusting candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Thrusting Indicator

To create an automatic indicator for `Thrusting` , call the `Thrusting` helper method from the `QCAlgorithm` class. The `Thrusting` method creates a `Thrusting` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class ThrustingAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Thrusting _thrusting;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _thrusting = CandlestickPatterns.Thrusting(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_thrusting.IsReady)
        {
            // The current value of _thrusting is represented by itself (_thrusting)
            // or _thrusting.Current.Value
            Plot("Thrusting", "thrusting", _thrusting);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `Thrusting` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class ThrustingAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Thrusting _thrusting;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _thrusting = new Thrusting();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _thrusting.Update(bar);

        if (_thrusting.IsReady)
        {
            // The current value of _thrusting is represented by itself (_thrusting)
            // or _thrusting.Current.Value
            Plot("Thrusting", "thrusting", _thrusting);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `Thrusting` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class ThrustingAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Thrusting _thrusting;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _thrusting = CandlestickPatterns.Thrusting(_symbol);

        var indicatorHistory = IndicatorHistory(_thrusting, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_thrusting, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_thrusting, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Tristar

Introduction

Create a new Tristar candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using Tristar Indicator

To create an automatic indicator for `Tristar` , call the `Tristar` helper method from the `QCAlgorithm` class. The `Tristar` method creates a `Tristar` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class TristarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Tristar _tristar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tristar = CandlestickPatterns.Tristar(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_tristar.IsReady)
        {
            // The current value of _tristar is represented by itself (_tristar)
            // or _tristar.Current.Value
            Plot("Tristar", "tristar", _tristar);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `Tristar` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class TristarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Tristar _tristar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tristar = new Tristar();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _tristar.Update(bar);

        if (_tristar.IsReady)
        {
            // The current value of _tristar is represented by itself (_tristar)
            // or _tristar.Current.Value
            Plot("Tristar", "tristar", _tristar);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `Tristar` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class TristarAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Tristar _tristar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tristar = CandlestickPatterns.Tristar(_symbol);

        var indicatorHistory = IndicatorHistory(_tristar, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_tristar, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_tristar, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Two Crows

Introduction

Create a new Two Crows candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using TwoCrows Indicator

To create an automatic indicator for `TwoCrows` , call the `TwoCrows` helper method from the `QCAlgorithm` class. The `TwoCrows` method creates a `TwoCrows` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class TwoCrowsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TwoCrows _twoCrows;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _twoCrows = CandlestickPatterns.TwoCrows(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_twoCrows.IsReady)
        {
            // The current value of _twoCrows is represented by itself (_twoCrows)
            // or _twoCrows.Current.Value
            Plot("TwoCrows", "twoCrows", _twoCrows);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `TwoCrows` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class TwoCrowsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TwoCrows _twoCrows;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _twoCrows = new TwoCrows();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _twoCrows.Update(bar);

        if (_twoCrows.IsReady)
        {
            // The current value of _twoCrows is represented by itself (_twoCrows)
            // or _twoCrows.Current.Value
            Plot("TwoCrows", "twoCrows", _twoCrows);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `TwoCrows` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class TwoCrowsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TwoCrows _twoCrows;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _twoCrows = CandlestickPatterns.TwoCrows(_symbol);

        var indicatorHistory = IndicatorHistory(_twoCrows, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_twoCrows, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_twoCrows, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Unique Three River

Introduction

Create a new Unique Three River candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using UniqueThreeRiver Indicator

To create an automatic indicator for `UniqueThreeRiver` , call the `UniqueThreeRiver` helper method from the `QCAlgorithm` class. The `UniqueThreeRiver` method creates a `UniqueThreeRiver` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class UniqueThreeRiverAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private UniqueThreeRiver _uniqueThreeRiver;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _uniqueThreeRiver = CandlestickPatterns.UniqueThreeRiver(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_uniqueThreeRiver.IsReady)
        {
            // The current value of _uniqueThreeRiver is represented by itself (_uniqueThreeRiver)
            // or _uniqueThreeRiver.Current.Value
            Plot("UniqueThreeRiver", "uniqueThreeRiver", _uniqueThreeRiver);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `UniqueThreeRiver` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class UniqueThreeRiverAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private UniqueThreeRiver _uniqueThreeRiver;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _uniqueThreeRiver = new UniqueThreeRiver();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _uniqueThreeRiver.Update(bar);

        if (_uniqueThreeRiver.IsReady)
        {
            // The current value of _uniqueThreeRiver is represented by itself (_uniqueThreeRiver)
            // or _uniqueThreeRiver.Current.Value
            Plot("UniqueThreeRiver", "uniqueThreeRiver", _uniqueThreeRiver);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `UniqueThreeRiver` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class UniqueThreeRiverAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private UniqueThreeRiver _uniqueThreeRiver;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _uniqueThreeRiver = CandlestickPatterns.UniqueThreeRiver(_symbol);

        var indicatorHistory = IndicatorHistory(_uniqueThreeRiver, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_uniqueThreeRiver, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_uniqueThreeRiver, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Up Down Gap Three Methods

Introduction

Create a new Up Down Gap Three Methods candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using UpDownGapThreeMethods Indicator

To create an automatic indicator for `UpDownGapThreeMethods` , call the `UpDownGapThreeMethods` helper method from the `QCAlgorithm` class. The `UpDownGapThreeMethods` method creates a `UpDownGapThreeMethods` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class UpDownGapThreeMethodsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private UpDownGapThreeMethods _upDownGapThreeMethods;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _upDownGapThreeMethods = CandlestickPatterns.UpDownGapThreeMethods(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_upDownGapThreeMethods.IsReady)
        {
            // The current value of _upDownGapThreeMethods is represented by itself (_upDownGapThreeMethods)
            // or _upDownGapThreeMethods.Current.Value
            Plot("UpDownGapThreeMethods", "upDownGapThreeMethods", _upDownGapThreeMethods);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `UpDownGapThreeMethods` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class UpDownGapThreeMethodsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private UpDownGapThreeMethods _upDownGapThreeMethods;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _upDownGapThreeMethods = new UpDownGapThreeMethods();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _upDownGapThreeMethods.Update(bar);

        if (_upDownGapThreeMethods.IsReady)
        {
            // The current value of _upDownGapThreeMethods is represented by itself (_upDownGapThreeMethods)
            // or _upDownGapThreeMethods.Current.Value
            Plot("UpDownGapThreeMethods", "upDownGapThreeMethods", _upDownGapThreeMethods);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `UpDownGapThreeMethods` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class UpDownGapThreeMethodsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private UpDownGapThreeMethods _upDownGapThreeMethods;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _upDownGapThreeMethods = CandlestickPatterns.UpDownGapThreeMethods(_symbol);

        var indicatorHistory = IndicatorHistory(_upDownGapThreeMethods, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_upDownGapThreeMethods, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_upDownGapThreeMethods, _symbol, new DateTime(2024, 7, 1),
new DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Upside Gap Two Crows

Introduction

Create a new Upside Gap Two Crows candlestick pattern to indicate the pattern's presence.

To view the implementation of this candlestick pattern, see the LEAN GitHub repository .

Using UpsideGapTwoCrows Indicator

To create an automatic indicator for `UpsideGapTwoCrows` , call the `UpsideGapTwoCrows` helper method from the `QCAlgorithm` class. The `UpsideGapTwoCrows` method creates a `UpsideGapTwoCrows` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class UpsideGapTwoCrowsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private UpsideGapTwoCrows _upsideGapTwoCrows;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _upsideGapTwoCrows = CandlestickPatterns.UpsideGapTwoCrows(_symbol);
    }

    public override void OnData(Slice data)
    {
        if (_upsideGapTwoCrows.IsReady)
        {
            // The current value of _upsideGapTwoCrows is represented by itself (_upsideGapTwoCrows)
            // or _upsideGapTwoCrows.Current.Value
            Plot("UpsideGapTwoCrows", "upsideGapTwoCrows", _upsideGapTwoCrows);
        }
    }
}
```

For more information about this method, see the CandlestickPatterns class .

You can manually create a `UpsideGapTwoCrows` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class UpsideGapTwoCrowsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private UpsideGapTwoCrows _upsideGapTwoCrows;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _upsideGapTwoCrows = new UpsideGapTwoCrows();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _upsideGapTwoCrows.Update(bar);

        if (_upsideGapTwoCrows.IsReady)
        {
            // The current value of _upsideGapTwoCrows is represented by itself (_upsideGapTwoCrows)
            // or _upsideGapTwoCrows.Current.Value
            Plot("UpsideGapTwoCrows", "upsideGapTwoCrows", _upsideGapTwoCrows);
        }
    }
}
```

For more information about this indicator, see its reference .

Indicator History

To get the historical data of the `UpsideGapTwoCrows` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class UpsideGapTwoCrowsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private UpsideGapTwoCrows _upsideGapTwoCrows;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _upsideGapTwoCrows = CandlestickPatterns.UpsideGapTwoCrows(_symbol);

        var indicatorHistory = IndicatorHistory(_upsideGapTwoCrows, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_upsideGapTwoCrows, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_upsideGapTwoCrows, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Absolute Price Oscillator

Introduction

This indicator computes the Absolute Price Oscillator (APO) The Absolute Price Oscillator is calculated using the following formula: APO[i] = FastMA[i] - SlowMA[i]

To view the implementation of this indicator, see the LEAN GitHub repository .

Using APO Indicator

To create an automatic indicator for `AbsolutePriceOscillator` , call the `APO` helper method from the `QCAlgorithm` class. The `APO` method creates a `AbsolutePriceOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class AbsolutePriceOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AbsolutePriceOscillator _apo;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _apo = APO(_symbol, 10, 20, MovingAverageType.Simple);
    }

    public override void OnData(Slice data)
    {

        if (_apo.IsReady)
        {
            // The current value of _apo is represented by itself (_apo)
            // or _apo.Current.Value
            Plot("AbsolutePriceOscillator", "apo", _apo);
            // Plot all properties of abands
            Plot("AbsolutePriceOscillator", "fast", _apo.Fast);
            Plot("AbsolutePriceOscillator", "slow", _apo.Slow);
            Plot("AbsolutePriceOscillator", "signal", _apo.Signal);
            Plot("AbsolutePriceOscillator", "histogram", _apo.Histogram);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `AbsolutePriceOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class AbsolutePriceOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AbsolutePriceOscillator _absolutepriceoscillator;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _absolutepriceoscillator = new AbsolutePriceOscillator(10, 20, MovingAverageType.Simple);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _absolutepriceoscillator.Update(bar.EndTime, bar.Close);

        if (_absolutepriceoscillator.IsReady)
        {
            // The current value of _absolutepriceoscillator is represented by itself (_absolutepriceoscillator)
            // or _absolutepriceoscillator.Current.Value
            Plot("AbsolutePriceOscillator", "absolutepriceoscillator", _absolutepriceoscillator);
            // Plot all properties of abands
            Plot("AbsolutePriceOscillator", "fast", _absolutepriceoscillator.Fast);
            Plot("AbsolutePriceOscillator", "slow", _absolutepriceoscillator.Slow);
            Plot("AbsolutePriceOscillator", "signal", _absolutepriceoscillator.Signal);
            Plot("AbsolutePriceOscillator", "histogram", _absolutepriceoscillator.Histogram);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `AbsolutePriceOscillator` indicator properties:

Indicator History

To get the historical data of the `AbsolutePriceOscillator` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security

subscription.

```C#
public class AbsolutePriceOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AbsolutePriceOscillator _apo;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _apo = APO(_symbol, 10, 20, MovingAverageType.Simple);

        var indicatorHistory = IndicatorHistory(_apo, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_apo, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_apo, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var fast = indicatorHistory.Select(x => ((dynamic)x).Fast).ToList();
        var slow = indicatorHistory.Select(x => ((dynamic)x).Slow).ToList();
        var signal = indicatorHistory.Select(x => ((dynamic)x).Signal).ToList();
        var histogram = indicatorHistory.Select(x => ((dynamic)x).Histogram).ToList();
    }
}
```

Supported Indicators

Acceleration Bands

Introduction

The Acceleration Bands created by Price Headley plots upper and lower envelope bands around a moving average.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ABANDS Indicator

To create an automatic indicator for `AccelerationBands` , call the `ABANDS` helper method from the `QCAlgorithm` class. The `ABANDS` method creates a `AccelerationBands` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class AccelerationBandsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AccelerationBands _abands;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _abands = ABANDS(_symbol, 10, 4m, MovingAverageType.Simple);
    }

    public override void OnData(Slice data)
    {

        if (_abands.IsReady)
        {
            // The current value of _abands is represented by itself (_abands)
            // or _abands.Current.Value
            Plot("AccelerationBands", "abands", _abands);
            // Plot all properties of abands
            Plot("AccelerationBands", "middleband", _abands.MiddleBand);
            Plot("AccelerationBands", "upperband", _abands.UpperBand);
            Plot("AccelerationBands", "lowerband", _abands.LowerBand);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `AccelerationBands` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

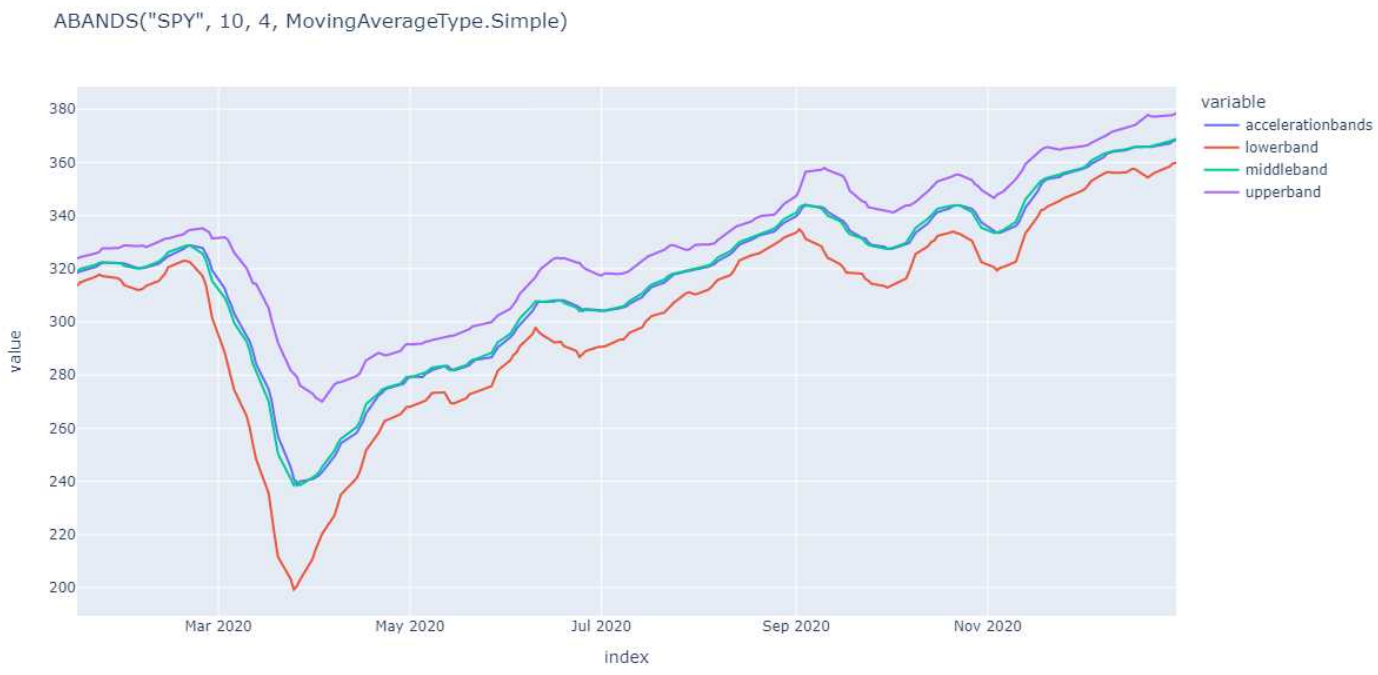
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class AccelerationBandsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AccelerationBands _accelerationbands;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _accelerationbands = new AccelerationBands(10, 4m, MovingAverageType.Simple);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _accelerationbands.Update(bar);

        if (_accelerationbands.IsReady)
        {
            // The current value of _accelerationbands is represented by itself (_accelerationbands)
            // or _accelerationbands.Current.Value
            Plot("AccelerationBands", "accelerationbands", _accelerationbands);
            // Plot all properties of abands
            Plot("AccelerationBands", "middleband", _accelerationbands.MiddleBand);
            Plot("AccelerationBands", "upperband", _accelerationbands.UpperBand);
            Plot("AccelerationBands", "lowerband", _accelerationbands.LowerBand);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `AccelerationBands` indicator properties:



Indicator History

To get the historical data of the `AccelerationBands` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class AccelerationBandsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AccelerationBands _abands;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _abands = ABANDS(_symbol, 10, 4m, MovingAverageType.Simple);

        var indicatorHistory = IndicatorHistory(_abands, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_abands, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_abands, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var middleBand = indicatorHistory.Select(x => ((dynamic)x).MiddleBand).ToList();
        var upperBand = indicatorHistory.Select(x => ((dynamic)x).UpperBand).ToList();
        var lowerBand = indicatorHistory.Select(x => ((dynamic)x).LowerBand).ToList();
    }
}
```

Supported Indicators

Accumulation Distribution

Introduction

This indicator computes the Accumulation/Distribution (AD) The Accumulation/Distribution is calculated using the following formula: AD = AD + ((Close - Low) - (High - Close)) / (High - Low) * Volume

To view the implementation of this indicator, see the LEAN GitHub repository .

Using AD Indicator

To create an automatic indicator for `AccumulationDistribution` , call the `AD` helper method from the `QCAlgorithm` class. The `AD` method creates a `AccumulationDistribution` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class AccumulationDistributionAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AccumulationDistribution _ad;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ad = AD(_symbol);
    }

    public override void OnData(Slice data)
    {

        if (_ad.IsReady)
        {
            // The current value of _ad is represented by itself (_ad)
            // or _ad.Current.Value
            Plot("AccumulationDistribution", "ad", _ad);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `AccumulationDistribution` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

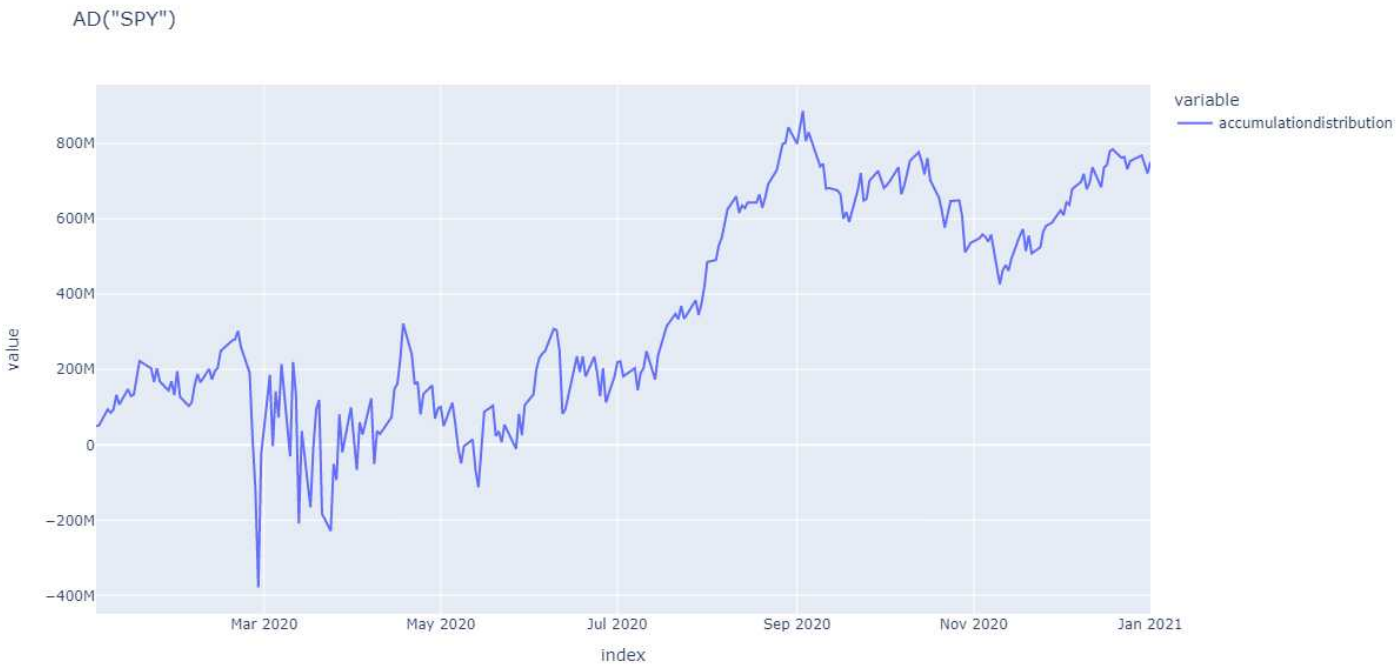
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class AccumulationDistributionAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AccumulationDistribution _accumulationdistribution;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _accumulationdistribution = new AccumulationDistribution();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _accumulationdistribution.Update(bar);

        if (_accumulationdistribution.IsReady)
        {
            // The current value of _accumulationdistribution is represented by itself (_accumulationdistribution)
            // or _accumulationdistribution.Current.Value
            Plot("AccumulationDistribution", "accumulationdistribution", _accumulationdistribution);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `AccumulationDistribution` indicator properties:



Indicator History

To get the historical data of the `AccumulationDistribution` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class AccumulationDistributionAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AccumulationDistribution _ad;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ad = AD(_symbol);

        var indicatorHistory = IndicatorHistory(_ad, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_ad, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_ad, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Accumulation Distribution Oscillator

Introduction

This indicator computes the Accumulation/Distribution Oscillator (ADOSC) The Accumulation/Distribution Oscillator is calculated using the following formula: ADOSC = EMA(fast,AD) - EMA(slow,AD)

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ADOSC Indicator

To create an automatic indicator for `AccumulationDistributionOscillator` , call the `ADOSC` helper method from the `QCAlgorithm` class. The `ADOSC` method creates a `AccumulationDistributionOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class AccumulationDistributionOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AccumulationDistributionOscillator _adosc;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _adosc = ADOSC(_symbol, 10, 20);
    }

    public override void OnData(Slice data)
    {

        if (_adosc.IsReady)
        {
            // The current value of _adosc is represented by itself (_adosc)
            // or _adosc.Current.Value
            Plot("AccumulationDistributionOscillator", "adosc", _adosc);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `AccumulationDistributionOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

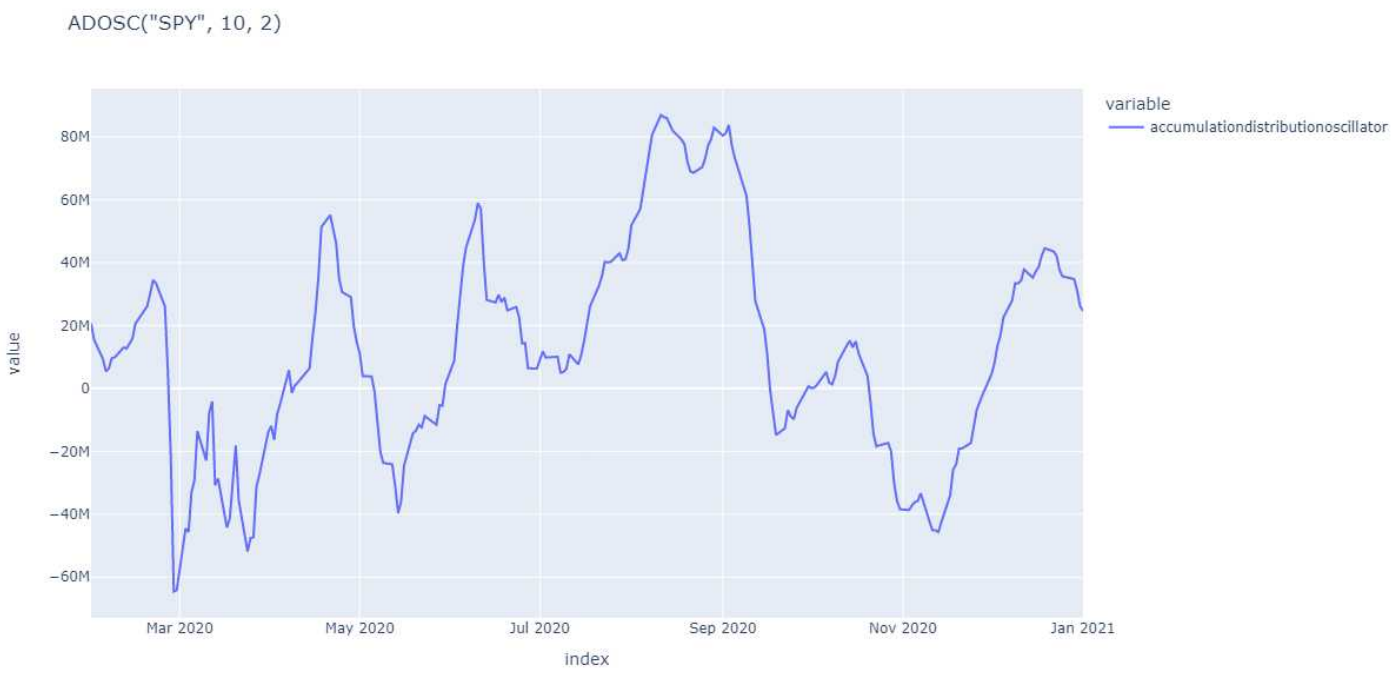
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class AccumulationDistributionOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AccumulationDistributionOscillator _accumulationdistributionoscillator;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _accumulationdistributionoscillator = new AccumulationDistributionOscillator(10, 20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _accumulationdistributionoscillator.Update(bar);

        if (_accumulationdistributionoscillator.IsReady)
        {
            // The current value of _accumulationdistributionoscillator is represented by itself
(_accumulationdistributionoscillator)
            // or _accumulationdistributionoscillator.Current.Value
            Plot("AccumulationDistributionOscillator", "accumulationdistributionoscillator",
_accumulationdistributionoscillator);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `AccumulationDistributionOscillator` indicator properties:



Indicator History

To get the historical data of the `AccumulationDistributionOscillator` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class AccumulationDistributionOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AccumulationDistributionOscillator _adosc;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _adosc = ADOSC(_symbol, 10, 20);

        var indicatorHistory = IndicatorHistory(_adosc, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_adosc, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_adosc, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Advance Decline Difference

Introduction

The Advance Decline Difference compute the difference between the number of stocks that closed higher and the number of stocks that closed lower than their previous day's closing prices.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ADDIFF Indicator

To create an automatic indicator for `AdvanceDeclineDifference` , call the `ADDIFF` helper method from the `QCAlgorithm` class. The `ADDIFF` method creates a `AdvanceDeclineDifference` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class AdvanceDeclineDifferenceAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private AdvanceDeclineDifference _addiff;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _addiff = ADDIFF([_symbol, _reference]);
    }

    public override void OnData(Slice data)
    {

        if (_addiff.IsReady)
        {
            // The current value of _addiff is represented by itself (_addiff)
            // or _addiff.Current.Value
            Plot("AdvanceDeclineDifference", "addiff", _addiff);
        }
    }
}
```

To create an automatic indicator for `AdvanceDeclineDifference` using universe constituents, call the `ADDIFF` helper method from the `QCAlgorithm` class. The `ADDIFF` method creates a `AdvanceDeclineDifference` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In this case, you should call the helper method in the `OnSecuritiesChanged` method.

```C#
public class AdvanceDeclineDifferenceAlgorithm : QCAlgorithm
{
    private Universe _universe;
    private AdvanceDeclineDifference _addiff;

    public override void Initialize()
    {
        SetStartDate(2024,1,1);
        SetEndDate(2024,12,31);
        UniverseSettings.Resolution = Resolution.Daily;
        UniverseSettings.Schedule.On(DateRules.MonthStart());
        _universe = AddUniverse(Universe.ETF("SPY"));
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        _addiff = ADDIFF(_universe.Selected);
    }

    public override void OnData(Slice data)
    {

        if (_addiff.IsReady)
        {
            // The current value of _addiff is represented by itself (_addiff)
            // or _addiff.Current.Value
            Plot("AdvanceDeclineDifference", "addiff", _addiff);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `AdvanceDeclineDifference` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

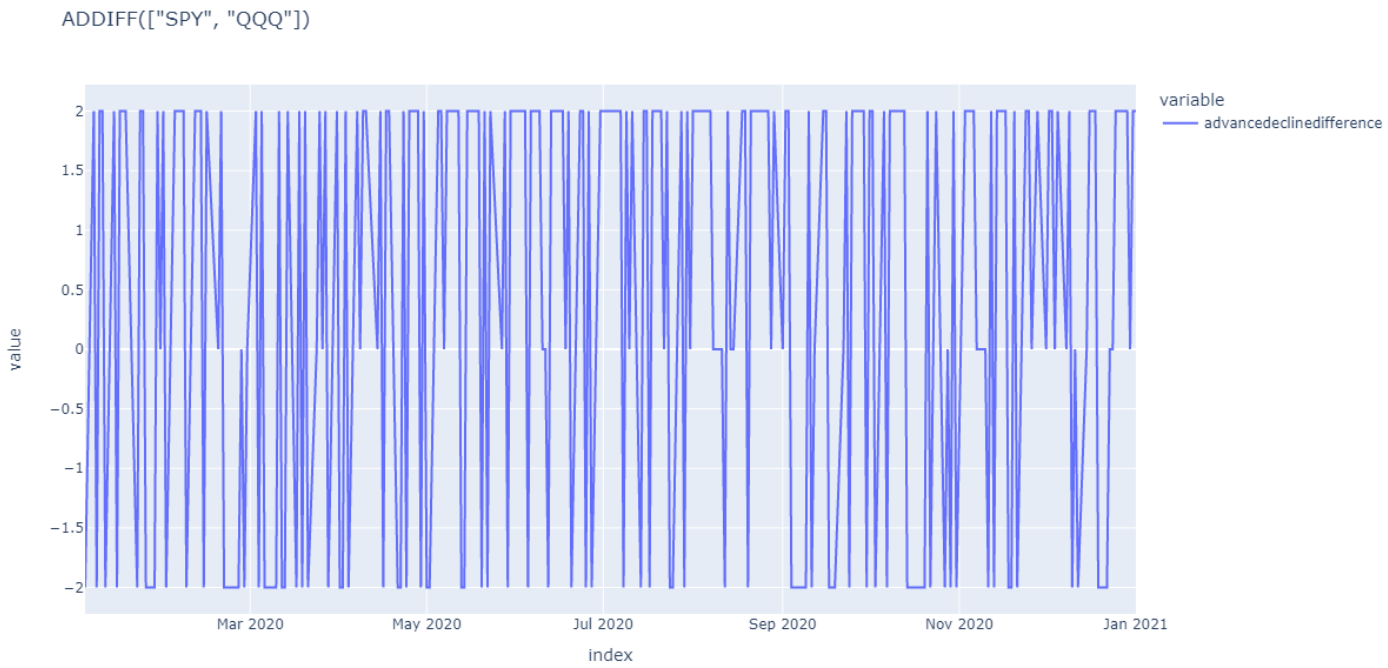
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class AdvanceDeclineDifferenceAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private AdvanceDeclineDifference _advancedeclinedifference;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _advancedeclinedifference = new AdvanceDeclineDifference();
        _advancedeclinedifference.Add(_symbol);
        _advancedeclinedifference.Add(_reference);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _advancedeclinedifference.Update(bar);
        if (data.Bars.TryGetValue(_reference, out bar))
            _advancedeclinedifference.Update(bar);

        if (_advancedeclinedifference.IsReady)
        {
            // The current value of _advancedeclinedifference is represented by itself (_advancedeclinedifference)
            // or _advancedeclinedifference.Current.Value
            Plot("AdvanceDeclineDifference", "advancedeclinedifference", _advancedeclinedifference);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `AdvanceDeclineDifference` indicator properties:



Indicator History

To get the historical data of the `AdvanceDeclineDifference` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class AdvanceDeclineDifferenceAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private AdvanceDeclineDifference _addiff;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _addiff = ADDIFF([_symbol, _reference]);

        var indicatorHistory = IndicatorHistory(_addiff, new[] { _symbol, _reference }, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_addiff, new[] { _symbol, _reference },
TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_addiff, new[] { _symbol, _reference }, new DateTime(2024,
7, 1), new DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Advance Decline Ratio

Introduction

The advance-decline ratio (ADR) compares the number of stocks that closed higher against the number of stocks that closed lower than their previous day's closing prices.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ADR Indicator

To create an automatic indicator for `AdvanceDeclineRatio` , call the `ADR` helper method from the `QCAlgorithm` class. The `ADR` method creates a `AdvanceDeclineRatio` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

C#

```
public class AdvanceDeclineRatioAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private AdvanceDeclineRatio _adr;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _adr = ADR(_symbol, _reference);
    }

    public override void OnData(Slice data)
    {

        if (_adr.IsReady)
        {
            // The current value of _adr is represented by itself (_adr)
            // or _adr.Current.Value
            Plot("AdvanceDeclineRatio", "adr", _adr);
        }
    }
}
```

To create an automatic indicator for `AdvanceDeclineRatio` using universe constituents, call the `ADR` helper method from the `QCAlgorithm` class. The `ADR` method creates a `AdvanceDeclineRatio` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In this case, you should call the helper method in the `OnSecuritiesChanged` method.

```C#
public class AdvanceDeclineRatioAlgorithm : QCAlgorithm
{
    private Universe _universe;
    private AdvanceDeclineRatio _adr;

    public override void Initialize()
    {
        SetStartDate(2024,1,1);
        SetEndDate(2024,12,31);
        UniverseSettings.Resolution = Resolution.Daily;
        UniverseSettings.Schedule.On(DateRules.MonthStart());
        _universe = AddUniverse(Universe.ETF("SPY"));
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        _adr = ADR(_universe.Selected);
    }

    public override void OnData(Slice data)
    {

        if (_adr.IsReady)
        {
            // The current value of _adr is represented by itself (_adr)
            // or _adr.Current.Value
            Plot("AdvanceDeclineRatio", "adr", _adr);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `AdvanceDeclineRatio` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

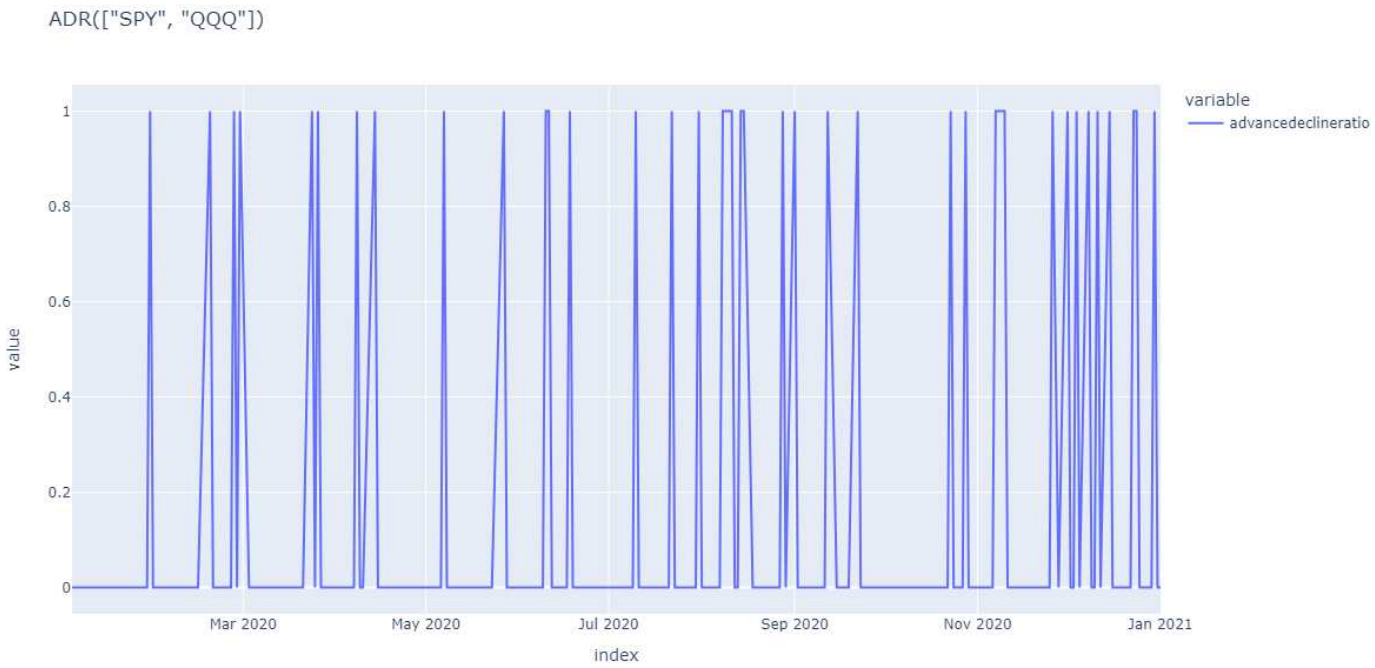
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class AdvanceDeclineRatioAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private AdvanceDeclineRatio _advancedeclineratio;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _advancedeclineratio = new AdvanceDeclineRatio();
        _advancedeclineratio.Add(_symbol);
        _advancedeclineratio.Add(_reference);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _advancedeclineratio.Update(bar);
        if (data.Bars.TryGetValue(_reference, out bar))
            _advancedeclineratio.Update(bar);

        if (_advancedeclineratio.IsReady)
        {
            // The current value of _advancedeclineratio is represented by itself (_advancedeclineratio)
            // or _advancedeclineratio.Current.Value
            Plot("AdvanceDeclineRatio", "advancedeclineratio", _advancedeclineratio);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `AdvanceDeclineRatio` indicator properties:



Indicator History

To get the historical data of the `AdvanceDeclineRatio` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class AdvanceDeclineRatioAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private AdvanceDeclineRatio _adr;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _adr = ADR([_symbol, _reference]);

        var indicatorHistory = IndicatorHistory(_adr, new[] { _symbol, _reference }, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_adr, new[] { _symbol, _reference }, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_adr, new[] { _symbol, _reference }, new DateTime(2024, 7,
1), new DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Advance Decline Volume Ratio

Introduction

The Advance Decline Volume Ratio is a Breadth indicator calculated as ratio of summary volume of advancing stocks to summary volume of declining stocks. AD Volume Ratio is used in technical analysis to see where the main trading activity is focused.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ADVR Indicator

To create an automatic indicator for `AdvanceDeclineVolumeRatio` , call the `ADVR` helper method from the `QCAlgorithm` class. The `ADVR` method creates a `AdvanceDeclineVolumeRatio` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

C#

```
public class AdvanceDeclineVolumeRatioAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private AdvanceDeclineVolumeRatio _advr;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _advr = ADVR([_symbol, _reference]);
    }

    public override void OnData(Slice data)
    {

        if (_advr.IsReady)
        {
            // The current value of _advr is represented by itself (_advr)
            // or _advr.Current.Value
            Plot("AdvanceDeclineVolumeRatio", "advr", _advr);
        }
    }
}
```

To create an automatic indicator for `AdvanceDeclineVolumeRatio` using universe constituents, call the `ADVR` helper method from the `QCAlgorithm` class. The `ADVR` method creates a `AdvanceDeclineVolumeRatio` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In this case, you should call the helper method in the `OnSecuritiesChanged` method.

```C#
public class AdvanceDeclineVolumeRatioAlgorithm : QCAlgorithm
{
    private Universe _universe;
    private AdvanceDeclineVolumeRatio _advr;

    public override void Initialize()
    {
        SetStartDate(2024,1,1);
        SetEndDate(2024,12,31);
        UniverseSettings.Resolution = Resolution.Daily;
        UniverseSettings.Schedule.On(DateRules.MonthStart());
        _universe = AddUniverse(Universe.ETF("SPY"));
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        _advr = ADVR(_universe.Selected);
    }

    public override void OnData(Slice data)
    {

        if (_advr.IsReady)
        {
            // The current value of _advr is represented by itself (_advr)
            // or _advr.Current.Value
            Plot("AdvanceDeclineVolumeRatio", "advr", _advr);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `AdvanceDeclineVolumeRatio` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

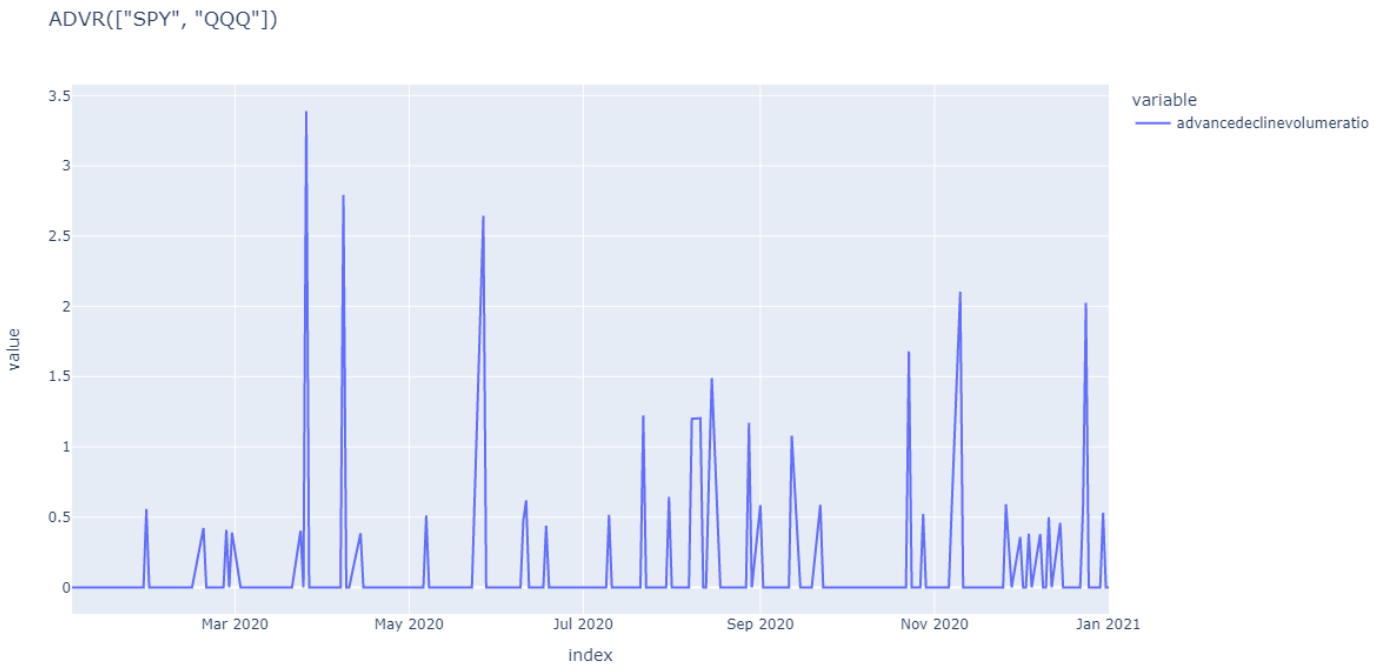
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class AdvanceDeclineVolumeRatioAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private AdvanceDeclineVolumeRatio _advancedeclinevolumeratio;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _advancedeclinevolumeratio = new AdvanceDeclineVolumeRatio();
        _advancedeclinevolumeratio.Add(_symbol);
        _advancedeclinevolumeratio.Add(_reference);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _advancedeclinevolumeratio.Update(bar);
        if (data.Bars.TryGetValue(_reference, out bar))
            _advancedeclinevolumeratio.Update(bar);

        if (_advancedeclinevolumeratio.IsReady)
        {
            // The current value of _advancedeclinevolumeratio is represented by itself (_advancedeclinevolumeratio)
            // or _advancedeclinevolumeratio.Current.Value
            Plot("AdvanceDeclineVolumeRatio", "advancedeclinevolumeratio", _advancedeclinevolumeratio);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `AdvanceDeclineVolumeRatio` indicator properties:



Indicator History

To get the historical data of the `AdvanceDeclineVolumeRatio` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class AdvanceDeclineVolumeRatioAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private AdvanceDeclineVolumeRatio _advr;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _advr = ADVR([_symbol, _reference]);

        var indicatorHistory = IndicatorHistory(_advr, new[] { _symbol, _reference }, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_advr, new[] { _symbol, _reference }, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_advr, new[] { _symbol, _reference }, new DateTime(2024, 7,
1), new DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Alpha

Introduction

In financial analysis, the Alpha indicator is used to measure the performance of an investment (such as a stock or ETF) relative to a benchmark index, often representing the broader market. Alpha indicates the excess return of the investment compared to the return of the benchmark index. The S P 500 index is frequently used as a benchmark in Alpha calculations to represent the overall market performance. Alpha is an essential tool for investors to understand the idiosyncratic returns of their investment that aren't caused by movement in the underlying benchmark.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using A Indicator

To create an automatic indicator for `Alpha` , call the `A` helper method from the `QCAlgorithm` class. The `A` method creates a `Alpha` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class AlphaAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private Alpha _a;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _a = A(_symbol, _reference, 20);
    }

    public override void OnData(Slice data)
    {

        if (_a.IsReady)
        {
            // The current value of _a is represented by itself (_a)
            // or _a.Current.Value
            Plot("Alpha", "a", _a);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `Alpha` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

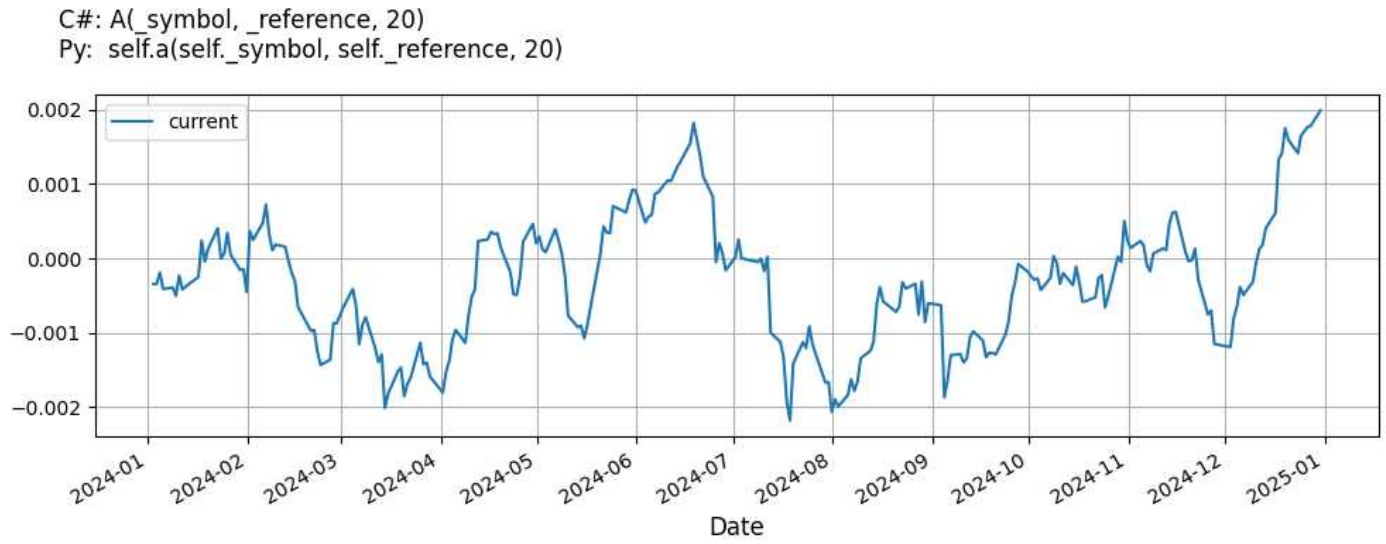
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class AlphaAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private Alpha _alpha;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _alpha = new Alpha(_symbol, _reference, 20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _alpha.Update(bar);
        if (data.Bars.TryGetValue(_reference, out bar))
            _alpha.Update(bar);

        if (_alpha.IsReady)
        {
            // The current value of _alpha is represented by itself (_alpha)
            // or _alpha.Current.Value
            Plot("Alpha", "alpha", _alpha);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `Alpha` indicator properties:



Indicator History

To get the historical data of the `Alpha` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class AlphaAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private Alpha _a;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _a = A(_symbol, _reference, 20);

        var indicatorHistory = IndicatorHistory(_a, new[] { _symbol, _reference }, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_a, new[] { _symbol, _reference }, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_a, new[] { _symbol, _reference }, new DateTime(2024, 7,
1), new DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Arms Index

Introduction

The Arms Index, also called the Short-Term Trading Index (TRIN) is a technical analysis indicator that compares the number of advancing and declining stocks (AD Ratio) to advancing and declining volume (AD volume).

To view the implementation of this indicator, see the LEAN GitHub repository .

Using TRIN Indicator

To create an automatic indicator for `ArmsIndex` , call the `TRIN` helper method from the `QCAlgorithm` class. The `TRIN` method creates a `ArmsIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class ArmsIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private ArmsIndex _trin;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _trin = TRIN([_symbol, _reference]);
    }

    public override void OnData(Slice data)
    {

        if (_trin.IsReady)
        {
            // The current value of _trin is represented by itself (_trin)
            // or _trin.Current.Value
            Plot("ArmsIndex", "trin", _trin);
            // Plot all properties of abands
            Plot("ArmsIndex", "adratio", _trin.ADRatio);
            Plot("ArmsIndex", "advratio", _trin.ADVRatio);
        }
    }
}
```

To create an automatic indicator for `ArmsIndex` using universe constituents, call the `TRIN` helper method from the `QCAlgorithm` class. The `TRIN` method creates a `ArmsIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In this case, you should call the helper method in the `OnSecuritiesChanged` method.

```C#
public class ArmsIndexAlgorithm : QCAlgorithm
{
    private Universe _universe;
    private ArmsIndex _trin;

    public override void Initialize()
    {
        SetStartDate(2024,1,1);
        SetEndDate(2024,12,31);
        UniverseSettings.Resolution = Resolution.Daily;
        UniverseSettings.Schedule.On(DateRules.MonthStart());
        _universe = AddUniverse(Universe.ETF("SPY"));
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        _trin = TRIN(_universe.Selected);
    }

    public override void OnData(Slice data)
    {

        if (_trin.IsReady)
        {
            // The current value of _trin is represented by itself (_trin)
            // or _trin.Current.Value
            Plot("ArmsIndex", "trin", _trin);
            // Plot all properties of abands
            Plot("ArmsIndex", "adratio", _trin.ADRatio);
            Plot("ArmsIndex", "advratio", _trin.ADVRatio);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `ArmsIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

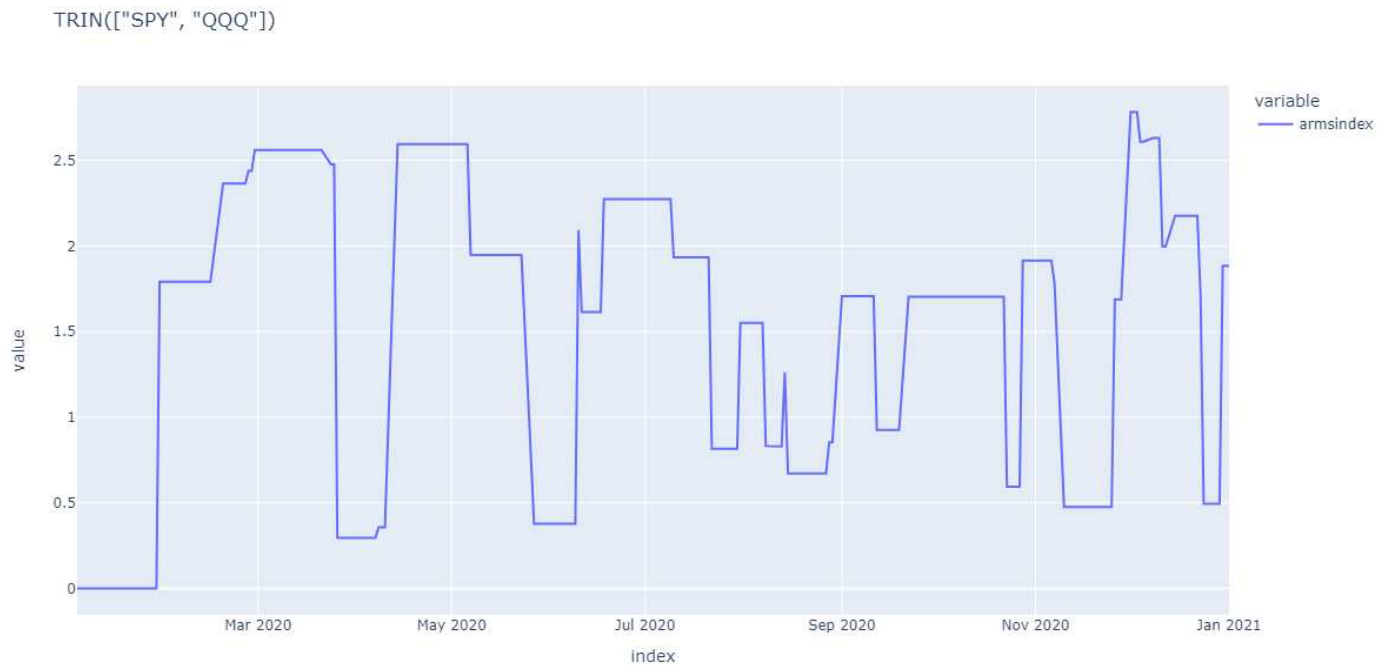
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ArmsIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private ArmsIndex _armsindex;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _armsindex = new ArmsIndex();
        _armsindex.Add(_symbol);
        _armsindex.Add(_reference);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _armsindex.Update(bar);
        if (data.Bars.TryGetValue(_reference, out bar))
            _armsindex.Update(bar);

        if (_armsindex.IsReady)
        {
            // The current value of _armsindex is represented by itself (_armsindex)
            // or _armsindex.Current.Value
            Plot("ArmsIndex", "armsindex", _armsindex);
            // Plot all properties of abands
            Plot("ArmsIndex", "adratio", _armsindex.ADRatio);
            Plot("ArmsIndex", "advratio", _armsindex.ADVRatio);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `ArmsIndex` indicator properties:



Indicator History

To get the historical data of the `ArmsIndex` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the

`IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class ArmsIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private ArmsIndex _trin;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _trin = TRIN([_symbol, _reference]);

        var indicatorHistory = IndicatorHistory(_trin, new[] { _symbol, _reference }, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_trin, new[] { _symbol, _reference }, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_trin, new[] { _symbol, _reference }, new DateTime(2024, 7,
1), new DateTime(2024, 7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var aDRatio = indicatorHistory.Select(x => ((dynamic)x).ADRatio).ToList();
        var aDVRatio = indicatorHistory.Select(x => ((dynamic)x).ADVRatio).ToList();
    }
}
```

Supported Indicators

Arnaud Legoux Moving Average

Introduction

Smooth and high sensitive moving Average. This moving average reduce lag of the information but still being smooth to reduce noises. Is a weighted moving average, which weights have a Normal shape; the parameters Sigma and Offset affect the kurtosis and skewness of the weights respectively. source

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ALMA Indicator

To create an automatic indicator for `ArnaudLegouxMovingAverage` , call the `ALMA` helper method from the `QCAlgorithm` class. The `ALMA` method creates a `ArnaudLegouxMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

C#

```
public class ArnaudLegouxMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ArnaudLegouxMovingAverage _alma;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _alma = ALMA(_symbol, 10, 6, 0.85m);
    }

    public override void OnData(Slice data)
    {

        if (_alma.IsReady)
        {
            // The current value of _alma is represented by itself (_alma)
            // or _alma.Current.Value
            Plot("ArnaudLegouxMovingAverage", "alma", _alma);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `ArnaudLegouxMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

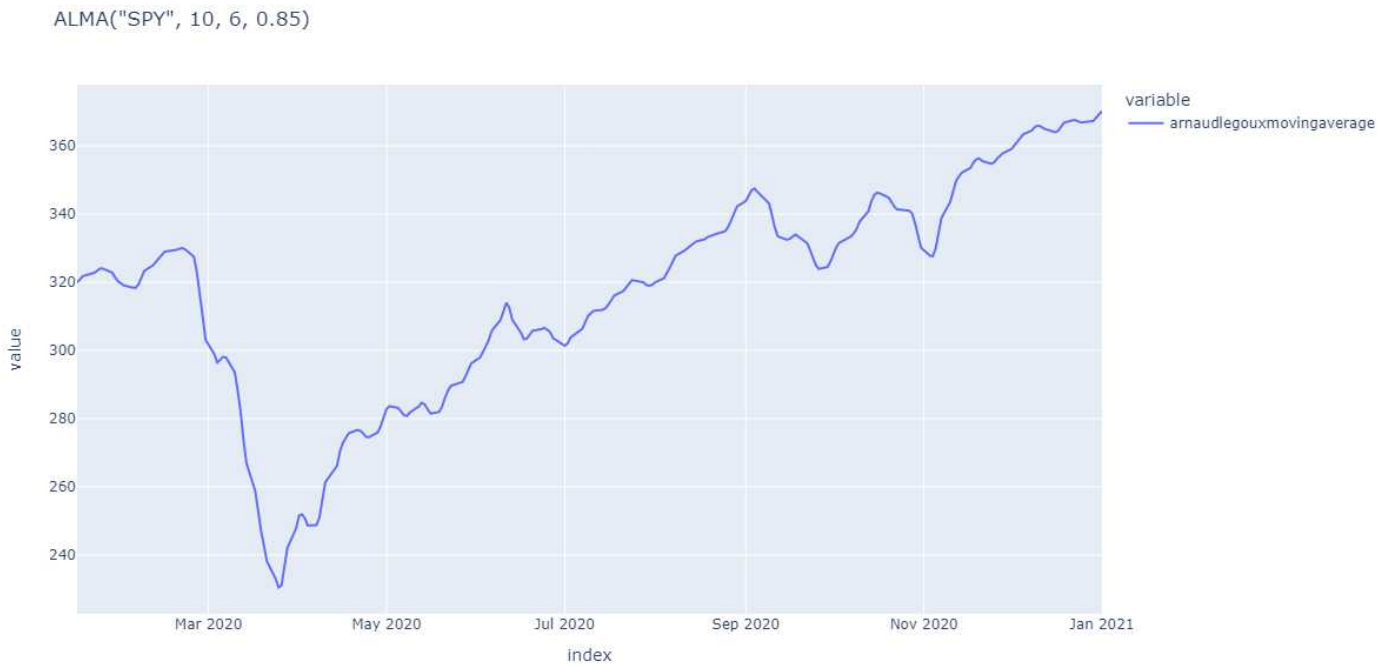
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ArnaudLegouxMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ArnaudLegouxMovingAverage _arnaudlegouxmovingaverage;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _arnaudlegouxmovingaverage = new ArnaudLegouxMovingAverage(10, 6, 0.85m);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _arnaudlegouxmovingaverage.Update(bar.EndTime, bar.Close);

        if (_arnaudlegouxmovingaverage.IsReady)
        {
            // The current value of _arnaudlegouxmovingaverage is represented by itself (_arnaudlegouxmovingaverage)
            // or _arnaudlegouxmovingaverage.Current.Value
            Plot("ArnaudLegouxMovingAverage", "arnaudlegouxmovingaverage", _arnaudlegouxmovingaverage);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `ArnaudLegouxMovingAverage` indicator properties:



Indicator History

To get the historical data of the `ArnaudLegouxMovingAverage` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class ArnaudLegouxMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ArnaudLegouxMovingAverage _alma;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _alma = ALMA(_symbol, 10, 6, 0.85m);

        var indicatorHistory = IndicatorHistory(_alma, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_alma, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_alma, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Aroon Oscillator

Introduction

The Aroon Oscillator is the difference between AroonUp and AroonDown. The value of this indicator fluctuates between -100 and +100. An upward trend bias is present when the oscillator is positive, and a negative trend bias is present when the oscillator is negative. AroonUp/Down values over 75 identify strong trends in their respective direction.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using AROON Indicator

To create an automatic indicator for `AroonOscillator` , call the `AROON` helper method from the `QCAlgorithm` class. The `AROON` method creates a `AroonOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class AroonOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AroonOscillator _aroon;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _aroon = AROON(_symbol, 10, 20);
    }

    public override void OnData(Slice data)
    {

        if (_aroon.IsReady)
        {
            // The current value of _aroon is represented by itself (_aroon)
            // or _aroon.Current.Value
            Plot("AroonOscillator", "aroon", _aroon);
            // Plot all properties of abands
            Plot("AroonOscillator", "aroonup", _aroon.AroonUp);
            Plot("AroonOscillator", "aroondown", _aroon.AroonDown);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `AroonOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

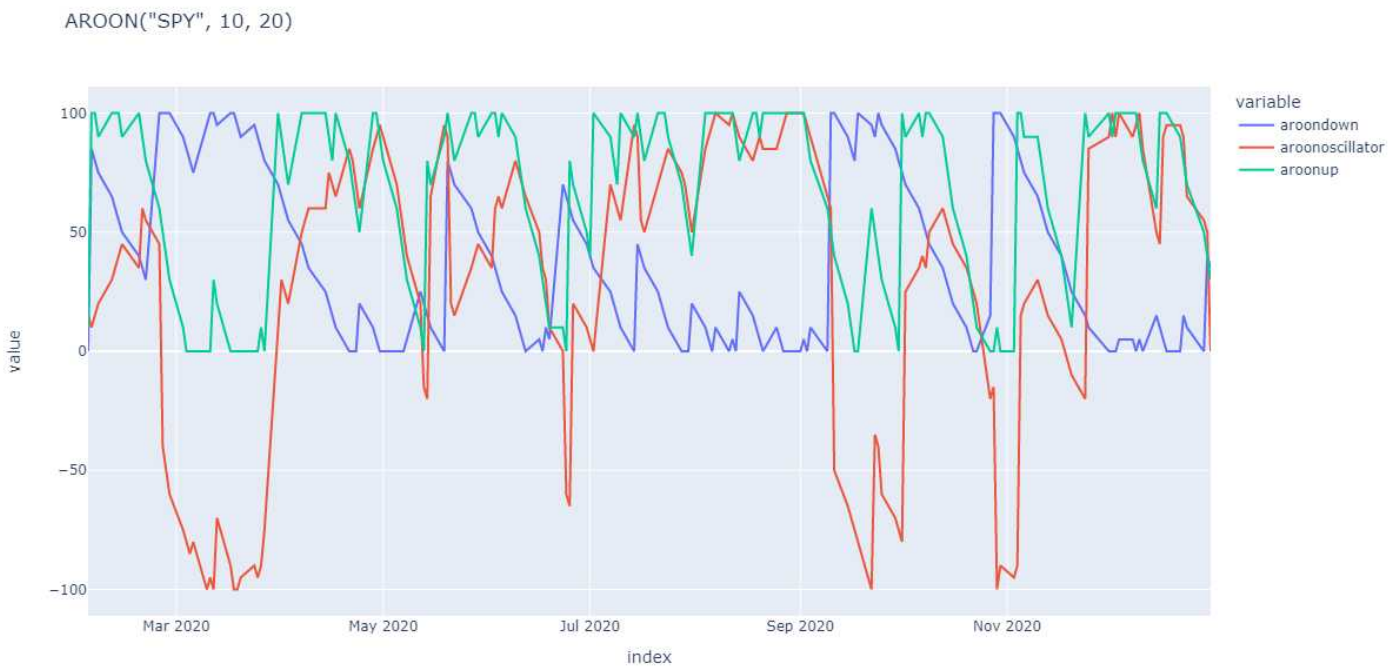
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class AroonOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AroonOscillator _aroonoscillator;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _aroonoscillator = new AroonOscillator(10, 20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _aroonoscillator.Update(bar);

        if (_aroonoscillator.IsReady)
        {
            // The current value of _aroonoscillator is represented by itself (_aroonoscillator)
            // or _aroonoscillator.Current.Value
            Plot("AroonOscillator", "aroonoscillator", _aroonoscillator);
            // Plot all properties of abands
            Plot("AroonOscillator", "aroonup", _aroonoscillator.AroonUp);
            Plot("AroonOscillator", "aroondown", _aroonoscillator.AroonDown);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `AroonOscillator` indicator properties:



Indicator History

To get the historical data of the `AroonOscillator` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class AroonOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AroonOscillator _aroon;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _aroon = AROON(_symbol, 10, 20);

        var indicatorHistory = IndicatorHistory(_aroon, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_aroon, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_aroon, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var aroonUp = indicatorHistory.Select(x => ((dynamic)x).AroonUp).ToList();
        var aroonDown = indicatorHistory.Select(x => ((dynamic)x).AroonDown).ToList();
    }
}
```

Supported Indicators

Augen Price Spike

Introduction

The Augen Price Spike indicator is an indicator that measures price changes in terms of standard deviations. In the book, The Volatility Edge in Options Trading, Jeff Augen describes a method for tracking absolute price changes in terms of recent volatility, using the standard deviation. length = x closes = closeArray closes1 = closeArray shifted right by 1 closes2 = closeArray shifted right by 2 closeLog = np.log(np.divide(closes1, closes2)) SDev = np.std(closeLog) m = SDev * closes1[-1] spike = (closes[-1]-closes1[-1])/m return spike Augen Price Spike from TradingView https://www.tradingview.com/script/fC7Pn2X2-Price-Spike-Jeff-Augen/

To view the implementation of this indicator, see the LEAN GitHub repository .

Using APS Indicator

To create an automatic indicator for `AugenPriceSpike` , call the `APS` helper method from the `QCAlgorithm` class. The `APS` method creates a `AugenPriceSpike` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class AugenPriceSpikeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AugenPriceSpike _aps;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _aps = APS(_symbol, 3);
    }

    public override void OnData(Slice data)
    {

        if (_aps.IsReady)
        {
            // The current value of _aps is represented by itself (_aps)
            // or _aps.Current.Value
            Plot("AugenPriceSpike", "aps", _aps);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `AugenPriceSpike` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

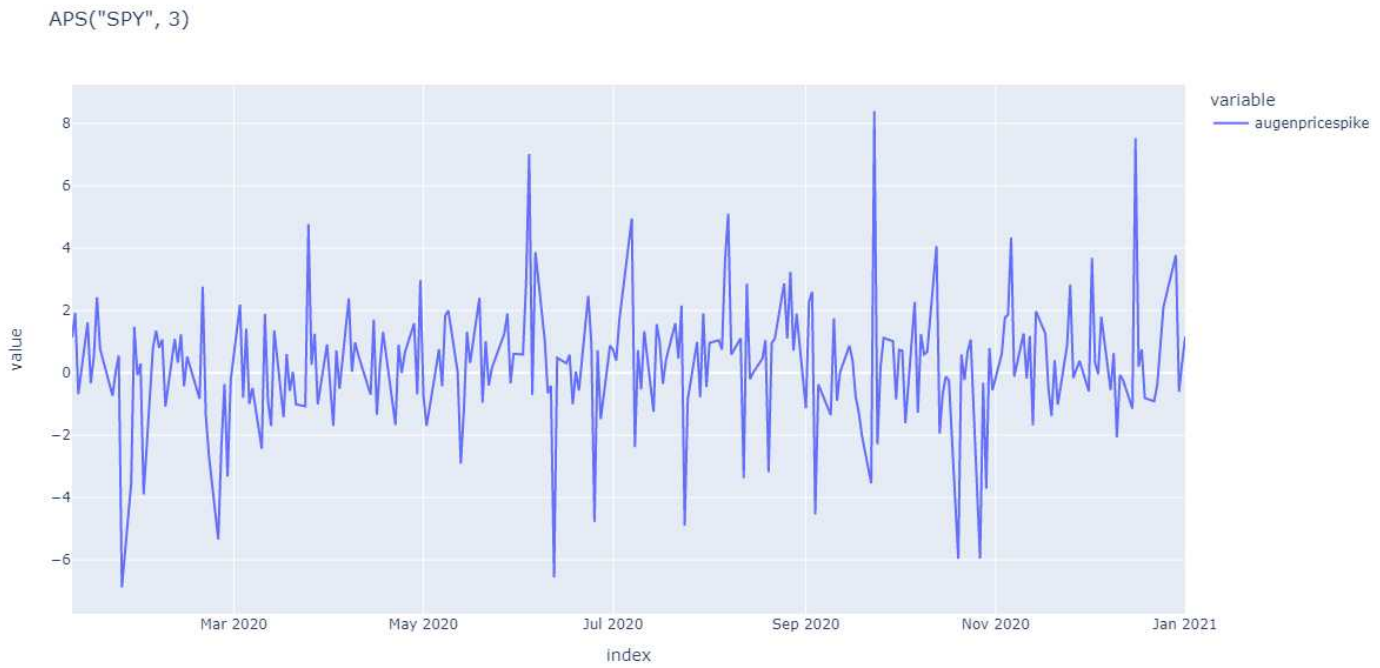
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class AugenPriceSpikeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AugenPriceSpike _augenpricespike;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _augenpricespike = new AugenPriceSpike(3);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _augenpricespike.Update(bar.EndTime, bar.Close);

        if (_augenpricespike.IsReady)
        {
            // The current value of _augenpricespike is represented by itself (_augenpricespike)
            // or _augenpricespike.Current.Value
            Plot("AugenPriceSpike", "augenpricespike", _augenpricespike);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `AugenPriceSpike` indicator properties:



Indicator History

To get the historical data of the `AugenPriceSpike` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class AugenPriceSpikeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AugenPriceSpike _aps;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _aps = APS(_symbol, 3);

        var indicatorHistory = IndicatorHistory(_aps, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_aps, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_aps, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Auto Regressive Integrated Moving Average

Introduction

An Autoregressive Intergrated Moving Average (ARIMA) is a time series model which can be used to describe a set of data. In particular,with X_t representing the series, the model assumes the data are of form (after differencing times): $X_t = c + \varepsilon_t + \Sigma_i \varphi_i X_{t-i} + \Sigma_i \theta_i \varepsilon_{t-i}$ where the first sum has an upper limit of and the second .

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ARIMA Indicator

To create an automatic indicator for `AutoRegressiveIntegratedMovingAverage` , call the `ARIMA` helper method from the `QCAlgorithm` class. The `ARIMA` method creates a `AutoRegressiveIntegratedMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class AutoRegressiveIntegratedMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AutoRegressiveIntegratedMovingAverage _arima;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _arima = ARIMA(_symbol, 1, 1, 1, 20);
    }

    public override void OnData(Slice data)
    {

        if (_arima.IsReady)
        {
            // The current value of _arima is represented by itself (_arima)
            // or _arima.Current.Value
            Plot("AutoRegressiveIntegratedMovingAverage", "arima", _arima);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `AutoRegressiveIntegratedMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

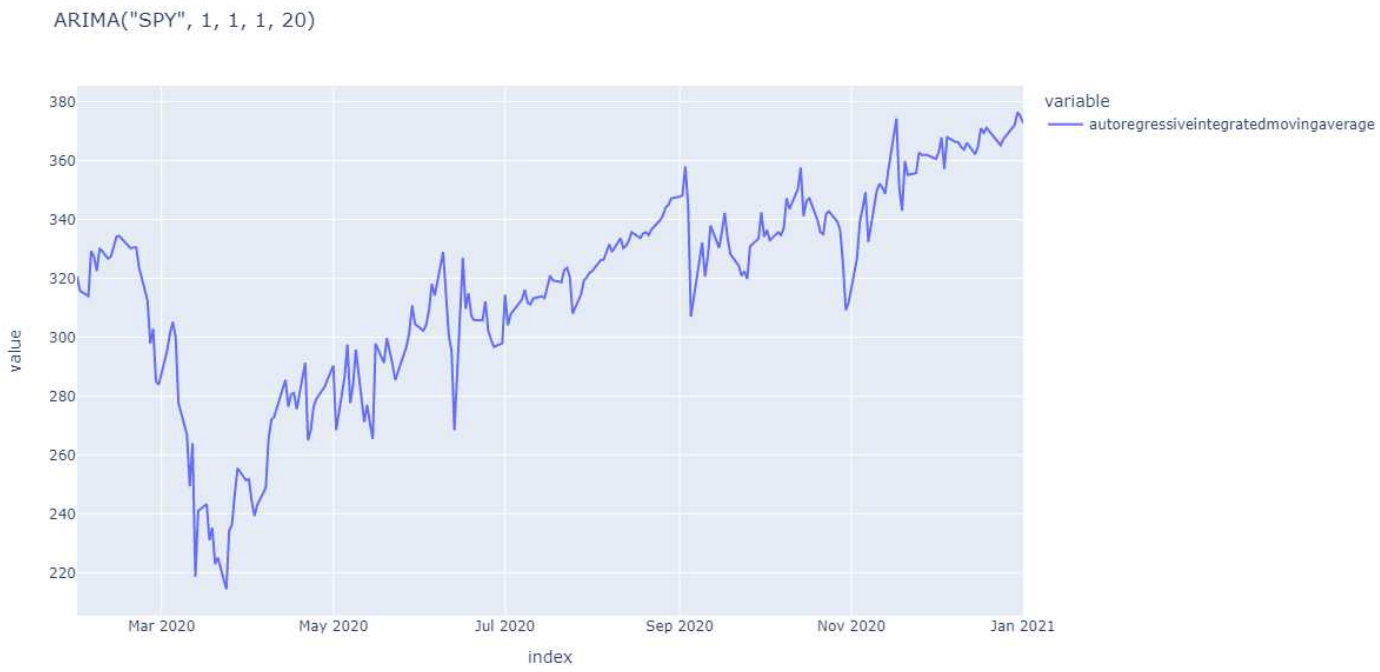
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class AutoRegressiveIntegratedMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AutoRegressiveIntegratedMovingAverage _autoregressiveintegratedmovingaverage;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _autoregressiveintegratedmovingaverage = new AutoRegressiveIntegratedMovingAverage(1, 1, 1, 20, true);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _autoregressiveintegratedmovingaverage.Update(bar.EndTime, bar.Close);

        if (_autoregressiveintegratedmovingaverage.IsReady)
        {
            // The current value of _autoregressiveintegratedmovingaverage is represented by itself
(_autoregressiveintegratedmovingaverage)
            // or _autoregressiveintegratedmovingaverage.Current.Value
            Plot("AutoRegressiveIntegratedMovingAverage", "autoregressiveintegratedmovingaverage",
_autoregressiveintegratedmovingaverage);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `AutoRegressiveIntegratedMovingAverage` indicator properties:



Indicator History

To get the historical data of the `AutoRegressiveIntegratedMovingAverage` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class AutoRegressiveIntegratedMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AutoRegressiveIntegratedMovingAverage _arima;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _arima = ARIMA(_symbol, 1, 1, 1, 20);

        var indicatorHistory = IndicatorHistory(_arima, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_arima, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_arima, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Average Directional Index

Introduction

This indicator computes Average Directional Index which measures trend strength without regard to trend direction. Firstly, it calculates the Directional Movement and the True Range value, and then the values are accumulated and smoothed using a custom smoothing method proposed by Wilder. For an n period smoothing, 1/n of each period's value is added to the total period. From these accumulated values we are therefore able to derived the 'Positive Directional Index' (+DI) and 'Negative Directional Index' (-DI) which is used to calculate the Average Directional Index. Computation source: https://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:average_directional_index_adx

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ADX Indicator

To create an automatic indicator for `AverageDirectionalIndex` , call the `ADX` helper method from the `QCAlgorithm` class. The `ADX` method creates a `AverageDirectionalIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class AverageDirectionalIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AverageDirectionalIndex _adx;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _adx = ADX(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_adx.IsReady)
        {
            // The current value of _adx is represented by itself (_adx)
            // or _adx.Current.Value
            Plot("AverageDirectionalIndex", "adx", _adx);
            // Plot all properties of abands
            Plot("AverageDirectionalIndex", "positivedirectionalindex", _adx.PositiveDirectionalIndex);
            Plot("AverageDirectionalIndex", "negativedirectionalindex", _adx.NegativeDirectionalIndex);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `AverageDirectionalIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class AverageDirectionalIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AverageDirectionalIndex _averagedirectionalindex;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _averagedirectionalindex = new AverageDirectionalIndex(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _averagedirectionalindex.Update(bar);

        if (_averagedirectionalindex.IsReady)
        {
            // The current value of _averagedirectionalindex is represented by itself (_averagedirectionalindex)
            // or _averagedirectionalindex.Current.Value
            Plot("AverageDirectionalIndex", "averagedirectionalindex", _averagedirectionalindex);
            // Plot all properties of abands
            Plot("AverageDirectionalIndex", "positivedirectionalindex",
_averagedirectionalindex.PositiveDirectionalIndex);
            Plot("AverageDirectionalIndex", "negativedirectionalindex",
_averagedirectionalindex.NegativeDirectionalIndex);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `AverageDirectionalIndex` indicator properties:

Indicator History

To get the historical data of the `AverageDirectionalIndex` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security

subscription.

```C#
public class AverageDirectionalIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AverageDirectionalIndex _adx;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _adx = ADX(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_adx, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_adx, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_adx, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var positiveDirectionalIndex = indicatorHistory.Select(x => ((dynamic)x).PositiveDirectionalIndex).ToList();
        var negativeDirectionalIndex = indicatorHistory.Select(x => ((dynamic)x).NegativeDirectionalIndex).ToList();
    }
}
```

Supported Indicators

Average Directional Movement Index Rating

Introduction

This indicator computes the Average Directional Movement Index Rating (ADXR). The Average Directional Movement Index Rating is calculated with the following formula: ADXR[i] = (ADX[i] + ADX[i - period + 1]) / 2

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ADXR Indicator

To create an automatic indicator for `AverageDirectionalMovementIndexRating` , call the `ADXR` helper method from the `QCAlgorithm` class. The `ADXR` method creates a `AverageDirectionalMovementIndexRating` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class AverageDirectionalMovementIndexRatingAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AverageDirectionalMovementIndexRating _adxr;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _adxr = ADXR(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_adxr.IsReady)
        {
            // The current value of _adxr is represented by itself (_adxr)
            // or _adxr.Current.Value
            Plot("AverageDirectionalMovementIndexRating", "adxr", _adxr);
            // Plot all properties of abands
            Plot("AverageDirectionalMovementIndexRating", "adx", _adxr.ADX);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `AverageDirectionalMovementIndexRating` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

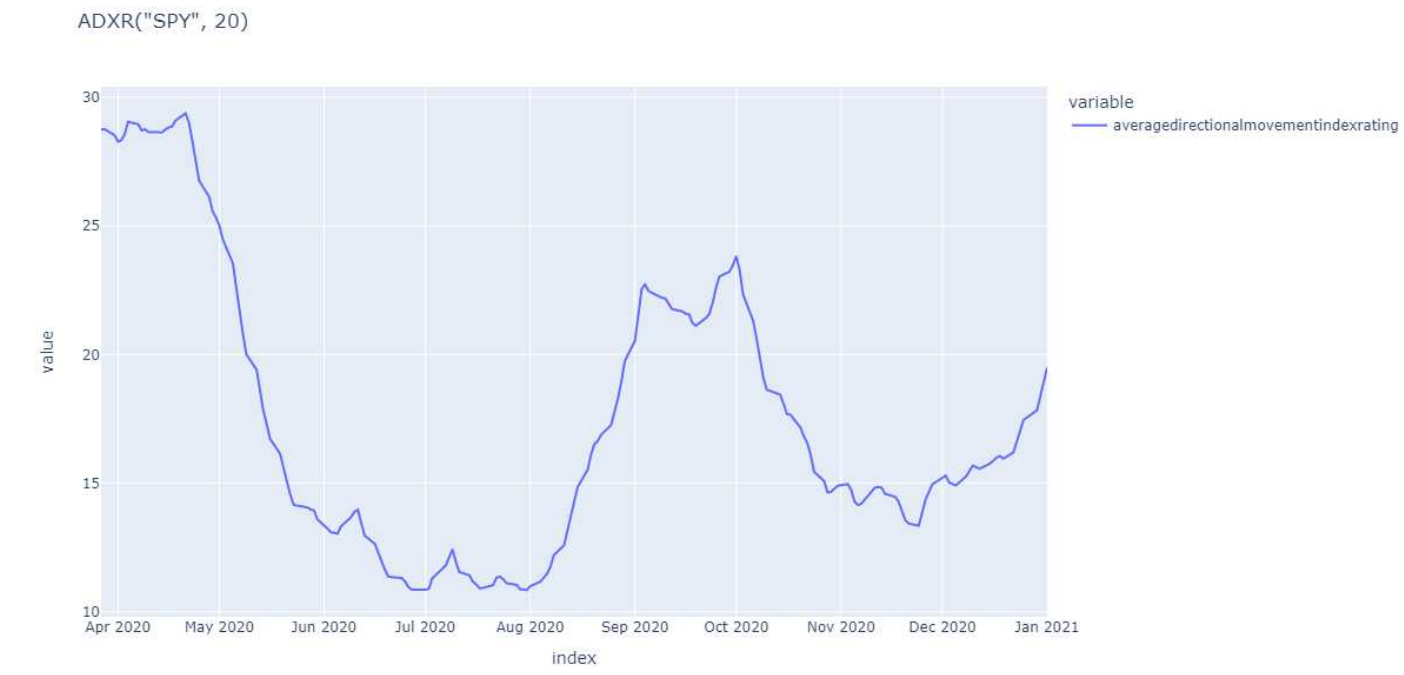
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class AverageDirectionalMovementIndexRatingAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AverageDirectionalMovementIndexRating _averagedirectionalmovementindexrating;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _averagedirectionalmovementindexrating = new AverageDirectionalMovementIndexRating(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _averagedirectionalmovementindexrating.Update(bar);

        if (_averagedirectionalmovementindexrating.IsReady)
        {
            // The current value of _averagedirectionalmovementindexrating is represented by itself
(_averagedirectionalmovementindexrating)
            // or _averagedirectionalmovementindexrating.Current.Value
            Plot("AverageDirectionalMovementIndexRating", "averagedirectionalmovementindexrating",
_averagedirectionalmovementindexrating);
            // Plot all properties of abands
            Plot("AverageDirectionalMovementIndexRating", "adx", _averagedirectionalmovementindexrating.ADX);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `AverageDirectionalMovementIndexRating` indicator properties:



Indicator History

To get the historical data of the `AverageDirectionalMovementIndexRating` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security

subscription.

```C#
public class AverageDirectionalMovementIndexRatingAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AverageDirectionalMovementIndexRating _adxr;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _adxr = ADXR(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_adxr, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_adxr, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_adxr, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var adx = indicatorHistory.Select(x => ((dynamic)x).ADX).ToList();
    }
}
```

Supported Indicators

Average Range

Introduction

This indicator represents the Average Range (AR) indicator, which calculates the average price range

To view the implementation of this indicator, see the LEAN GitHub repository .

Using AR Indicator

To create an automatic indicator for `AverageRange` , call the `AR` helper method from the `QCAlgorithm` class. The `AR` method creates a `AverageRange` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class AverageRangeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AverageRange _ar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ar = AR(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_ar.IsReady)
        {
            // The current value of _ar is represented by itself (_ar)
            // or _ar.Current.Value
            Plot("AverageRange", "ar", _ar);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `AverageRange` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

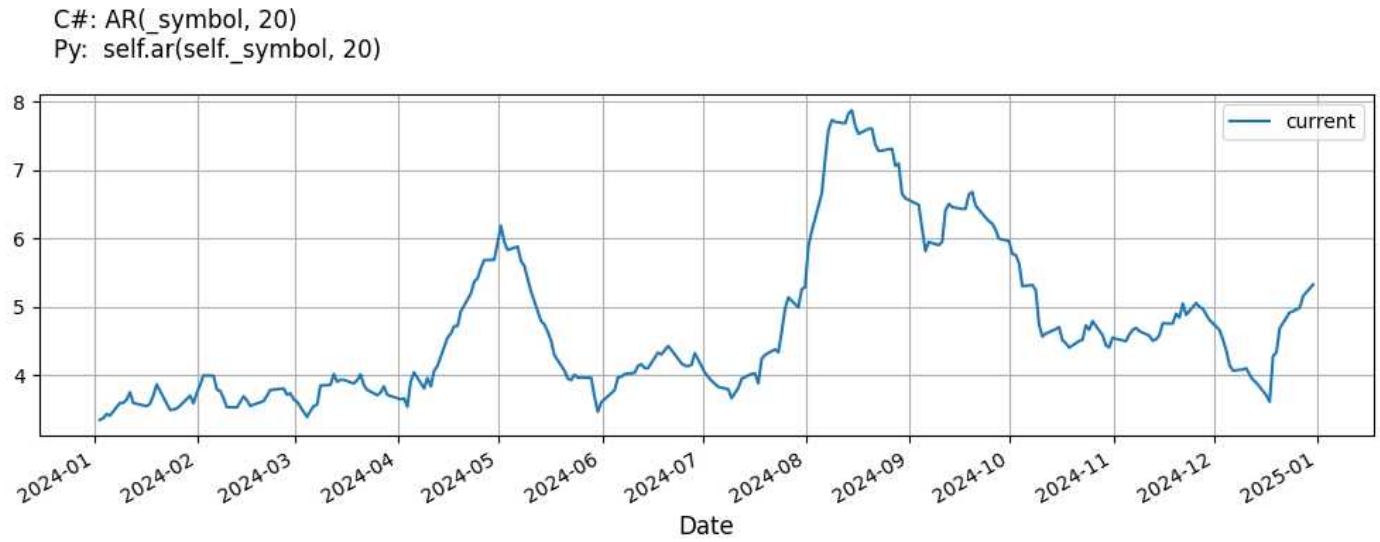
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class AverageRangeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AverageRange _averagerange;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _averagerange = new AverageRange(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _averagerange.Update(bar);

        if (_averagerange.IsReady)
        {
            // The current value of _averagerange is represented by itself (_averagerange)
            // or _averagerange.Current.Value
            Plot("AverageRange", "averagerange", _averagerange);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `AverageRange` indicator properties:



Indicator History

To get the historical data of the `AverageRange` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class AverageRangeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AverageRange _ar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ar = AR(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_ar, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_ar, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_ar, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Average True Range

Introduction

The AverageTrueRange indicator is a measure of volatility introduced by Welles Wilder in his book: New Concepts in Technical Trading Systems. This indicator computes the TrueRange and then smoothes the TrueRange over a given period. TrueRange is defined as the maximum of the following: High - Low ABS(High - PreviousClose) ABS(Low - PreviousClose)

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ATR Indicator

To create an automatic indicator for `AverageTrueRange` , call the `ATR` helper method from the `QCAlgorithm` class. The `ATR` method creates a `AverageTrueRange` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class AverageTrueRangeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AverageTrueRange _atr;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _atr = ATR(_symbol, 20, MovingAverageType.Simple);
    }

    public override void OnData(Slice data)
    {

        if (_atr.IsReady)
        {
            // The current value of _atr is represented by itself (_atr)
            // or _atr.Current.Value
            Plot("AverageTrueRange", "atr", _atr);
            // Plot all properties of abands
            Plot("AverageTrueRange", "truerange", _atr.TrueRange);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `AverageTrueRange` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

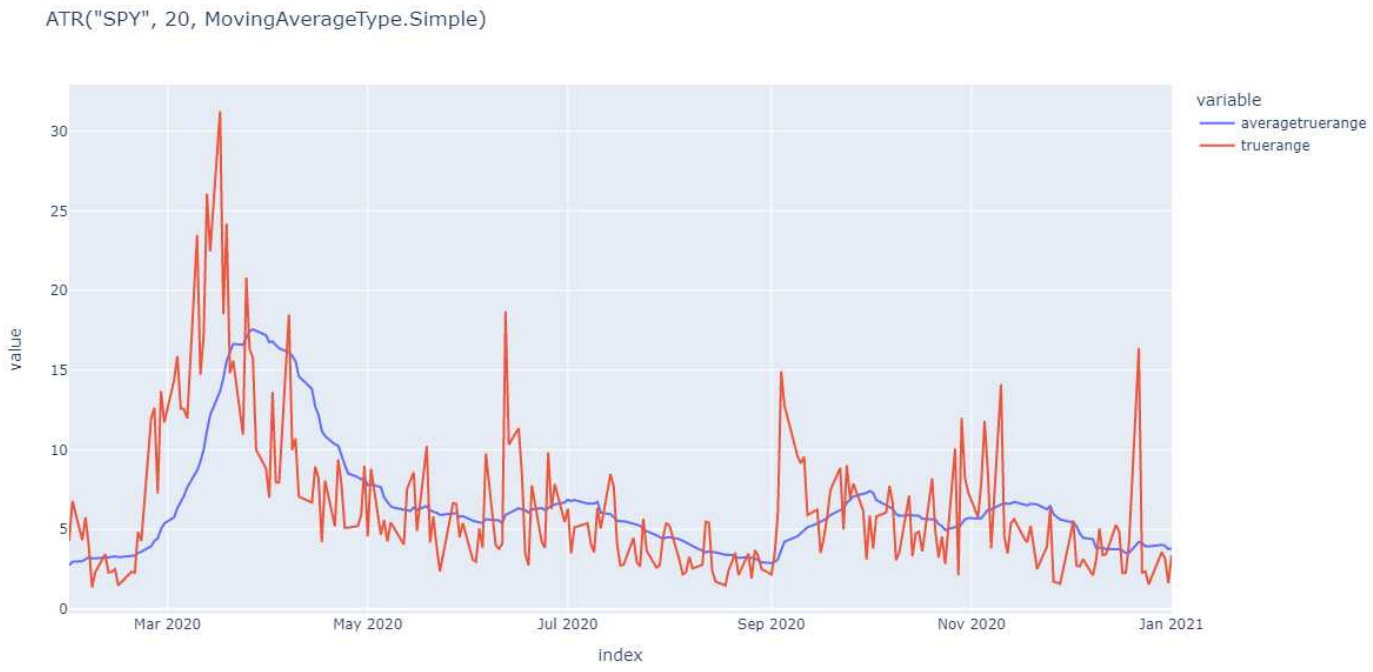
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class AverageTrueRangeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AverageTrueRange _averagetruerange;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _averagetruerange = new AverageTrueRange(20, MovingAverageType.Simple);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _averagetruerange.Update(bar);

        if (_averagetruerange.IsReady)
        {
            // The current value of _averagetruerange is represented by itself (_averagetruerange)
            // or _averagetruerange.Current.Value
            Plot("AverageTrueRange", "averagetruerange", _averagetruerange);
            // Plot all properties of abands
            Plot("AverageTrueRange", "truerange", _averagetruerange.TrueRange);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `AverageTrueRange` indicator properties:



Indicator History

To get the historical data of the `AverageTrueRange` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class AverageTrueRangeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AverageTrueRange _atr;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _atr = ATR(_symbol, 20, MovingAverageType.Simple);

        var indicatorHistory = IndicatorHistory(_atr, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_atr, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_atr, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var trueRange = indicatorHistory.Select(x => ((dynamic)x).TrueRange).ToList();
    }
}
```

Supported Indicators

Awesome Oscillator

Introduction

The Awesome Oscillator Indicator tracks the price midpoint-movement of a security. Specifically, AO = MAfast[(H+L)/2] - MAslow[(H+L)/2] where MAfast and MAslow denote simple moving averages wherein fast has a shorter period. https://www.barchart.com/education/technical-indicators/awesome_oscillator

To view the implementation of this indicator, see the LEAN GitHub repository .

Using AO Indicator

To create an automatic indicator for `AwesomeOscillator` , call the `AO` helper method from the `QCAlgorithm` class. The `AO` method creates a `AwesomeOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class AwesomeOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AwesomeOscillator _ao;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ao = AO(_symbol, 10, 20, MovingAverageType.Simple);
    }

    public override void OnData(Slice data)
    {

        if (_ao.IsReady)
        {
            // The current value of _ao is represented by itself (_ao)
            // or _ao.Current.Value
            Plot("AwesomeOscillator", "ao", _ao);
            // Plot all properties of abands
            Plot("AwesomeOscillator", "slowao", _ao.SlowAo);
            Plot("AwesomeOscillator", "fastao", _ao.FastAo);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `AwesomeOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

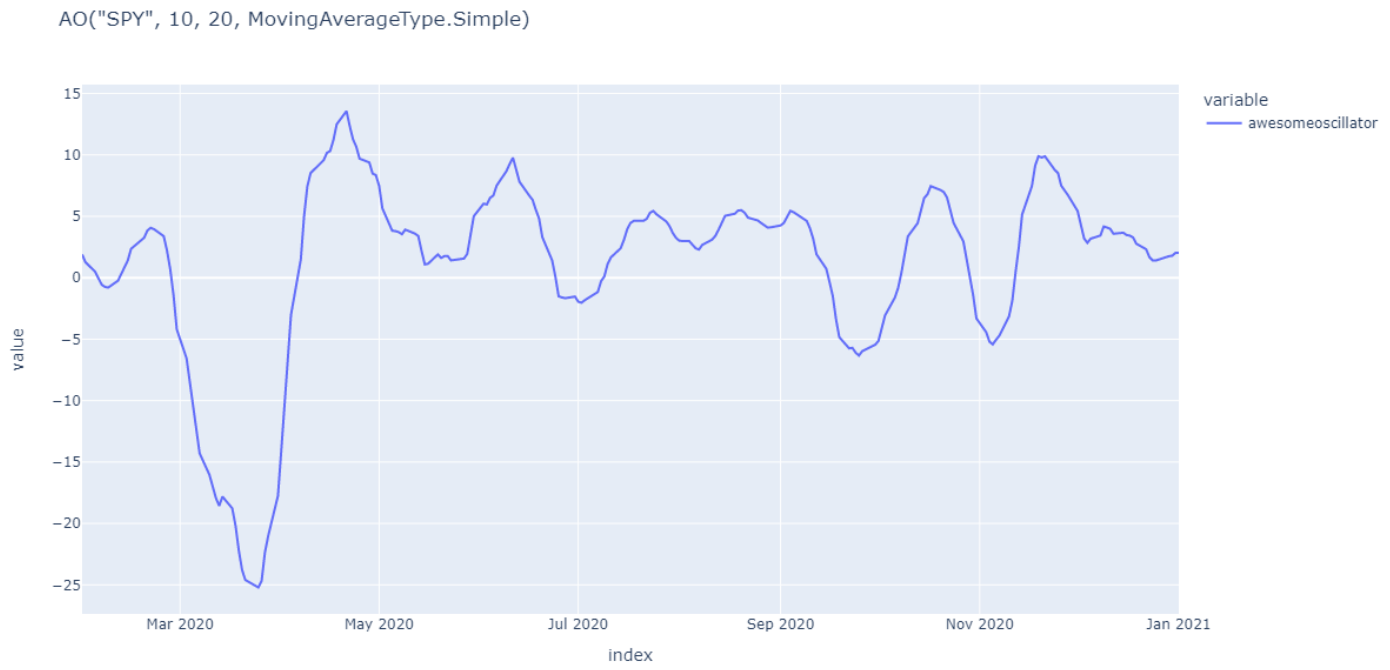
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class AwesomeOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AwesomeOscillator _awesomeoscillator;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _awesomeoscillator = new AwesomeOscillator(10, 20, MovingAverageType.Simple);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _awesomeoscillator.Update(bar);

        if (_awesomeoscillator.IsReady)
        {
            // The current value of _awesomeoscillator is represented by itself (_awesomeoscillator)
            // or _awesomeoscillator.Current.Value
            Plot("AwesomeOscillator", "awesomeoscillator", _awesomeoscillator);
            // Plot all properties of abands
            Plot("AwesomeOscillator", "slowao", _awesomeoscillator.SlowAo);
            Plot("AwesomeOscillator", "fastao", _awesomeoscillator.FastAo);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `AwesomeOscillator` indicator properties:



Indicator History

To get the historical data of the `AwesomeOscillator` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class AwesomeOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private AwesomeOscillator _ao;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ao = AO(_symbol, 10, 20, MovingAverageType.Simple);

        var indicatorHistory = IndicatorHistory(_ao, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_ao, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_ao, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var slowAo = indicatorHistory.Select(x => ((dynamic)x).SlowAo).ToList();
        var fastAo = indicatorHistory.Select(x => ((dynamic)x).FastAo).ToList();
    }
}
```

Supported Indicators

Balance Of Power

Introduction

This indicator computes the Balance Of Power (BOP). The Balance Of Power is calculated with the following formula: BOP = (Close - Open) / (High - Low)

To view the implementation of this indicator, see the LEAN GitHub repository .

Using BOP Indicator

To create an automatic indicator for `BalanceOfPower` , call the `BOP` helper method from the `QCAlgorithm` class. The `BOP` method creates a `BalanceOfPower` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class BalanceOfPowerAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private BalanceOfPower _bop;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _bop = BOP(_symbol);
    }

    public override void OnData(Slice data)
    {

        if (_bop.IsReady)
        {
            // The current value of _bop is represented by itself (_bop)
            // or _bop.Current.Value
            Plot("BalanceOfPower", "bop", _bop);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `BalanceOfPower` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

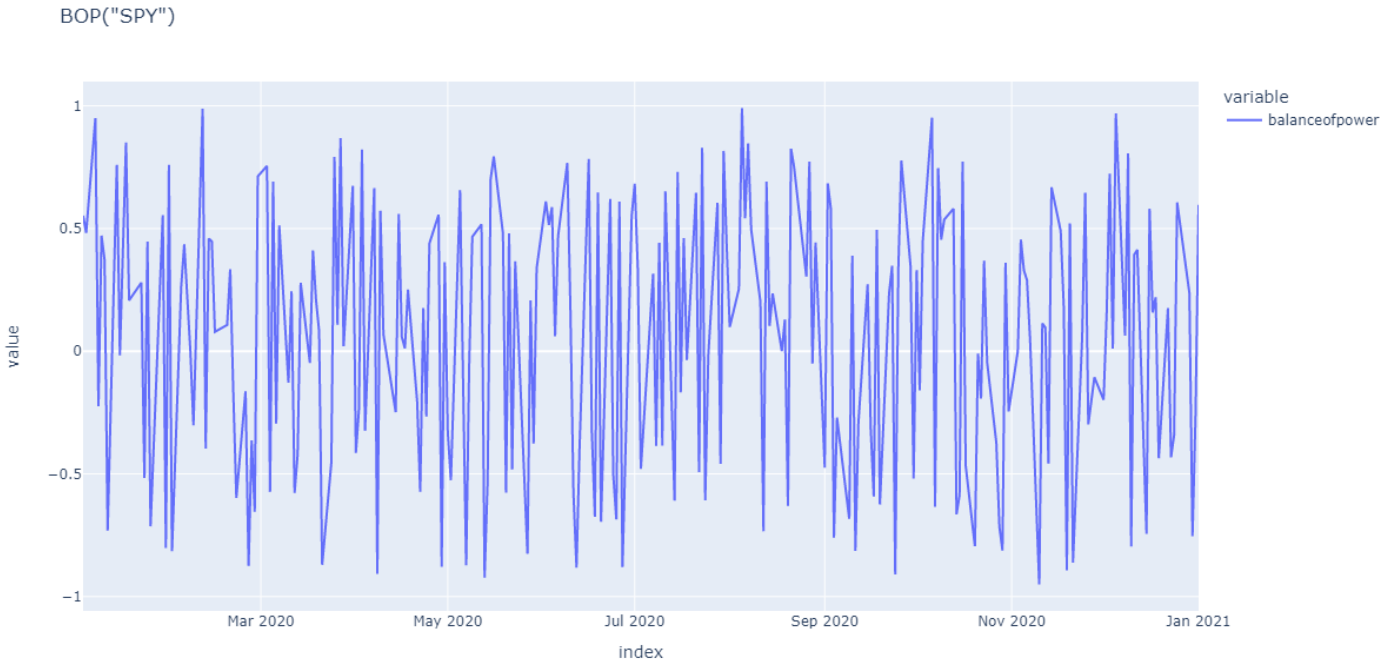
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class BalanceOfPowerAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private BalanceOfPower _balanceofpower;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _balanceofpower = new BalanceOfPower();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _balanceofpower.Update(bar);

        if (_balanceofpower.IsReady)
        {
            // The current value of _balanceofpower is represented by itself (_balanceofpower)
            // or _balanceofpower.Current.Value
            Plot("BalanceOfPower", "balanceofpower", _balanceofpower);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `BalanceOfPower` indicator properties:



Indicator History

To get the historical data of the `BalanceOfPower` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class BalanceOfPowerAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private BalanceOfPower _bop;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _bop = BOP(_symbol);

        var indicatorHistory = IndicatorHistory(_bop, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_bop, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_bop, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Beta

Introduction

In technical analysis Beta indicator is used to measure volatility or risk of a target (ETF) relative to the overall risk (volatility) of the reference (market indexes). The Beta indicators compares target's price movement to the movements of the indexes over the same period of time. It is common practice to use the SPX index as a benchmark of the overall reference market when it comes to Beta calculations. The indicator only updates when both assets have a price for a time step. When a bar is missing for one of the assets, the indicator value fills forward to improve the accuracy of the indicator.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using B Indicator

To create an automatic indicator for `Beta` , call the `B` helper method from the `QCAlgorithm` class. The `B` method creates a `Beta` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class BetaAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private Beta _b;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _b = B(_symbol, _reference, 20);
    }

    public override void OnData(Slice data)
    {

        if (_b.IsReady)
        {
            // The current value of _b is represented by itself (_b)
            // or _b.Current.Value
            Plot("Beta", "b", _b);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `Beta` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

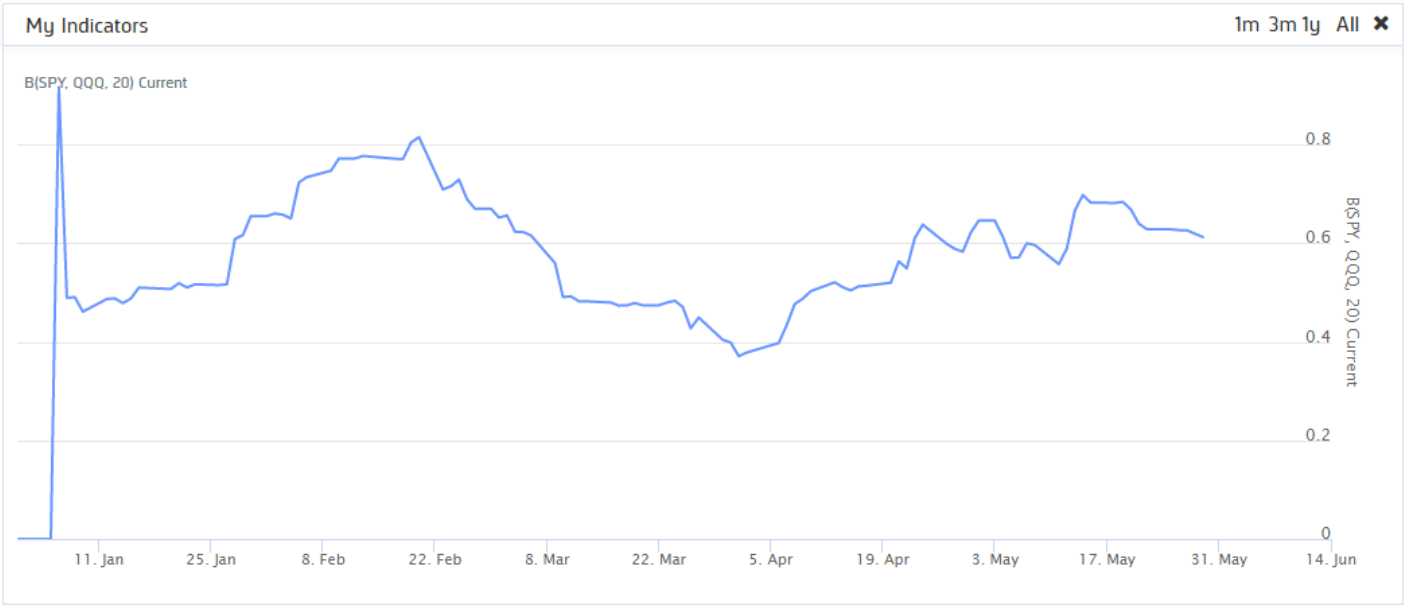
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class BetaAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private Beta _beta;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _beta = new Beta(_symbol, _reference, 20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _beta.Update(bar);
        if (data.Bars.TryGetValue(_reference, out bar))
            _beta.Update(bar);

        if (_beta.IsReady)
        {
            // The current value of _beta is represented by itself (_beta)
            // or _beta.Current.Value
            Plot("Beta", "beta", _beta);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `Beta` indicator properties:



Indicator History

To get the historical data of the `Beta` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class BetaAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private Beta _b;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _b = B(_symbol, _reference, 20);

        var indicatorHistory = IndicatorHistory(_b, new[] { _symbol, _reference }, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_b, new[] { _symbol, _reference }, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_b, new[] { _symbol, _reference }, new DateTime(2024, 7,
1), new DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Bollinger Bands

Introduction

This indicator creates a moving average (middle band) with an upper band and lower band fixed at k standard deviations above and below the moving average.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using BB Indicator

To create an automatic indicator for `BollingerBands` , call the `BB` helper method from the `QCAlgorithm` class. The `BB` method creates a `BollingerBands` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class BollingerBandsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private BollingerBands _bb;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _bb = BB(_symbol, 30, 2m);
    }

    public override void OnData(Slice data)
    {

        if (_bb.IsReady)
        {
            // The current value of _bb is represented by itself (_bb)
            // or _bb.Current.Value
            Plot("BollingerBands", "bb", _bb);
            // Plot all properties of abands
            Plot("BollingerBands", "standarddeviation", _bb.StandardDeviation);
            Plot("BollingerBands", "middleband", _bb.MiddleBand);
            Plot("BollingerBands", "upperband", _bb.UpperBand);
            Plot("BollingerBands", "lowerband", _bb.LowerBand);
            Plot("BollingerBands", "bandwidth", _bb.BandWidth);
            Plot("BollingerBands", "percentb", _bb.PercentB);
            Plot("BollingerBands", "price", _bb.Price);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `BollingerBands` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class BollingerBandsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private BollingerBands _bollingerbands;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _bollingerbands = new BollingerBands(30, 2m);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _bollingerbands.Update(bar.EndTime, bar.Close);

        if (_bollingerbands.IsReady)
        {
            // The current value of _bollingerbands is represented by itself (_bollingerbands)
            // or _bollingerbands.Current.Value
            Plot("BollingerBands", "bollingerbands", _bollingerbands);
            // Plot all properties of abands
            Plot("BollingerBands", "standarddeviation", _bollingerbands.StandardDeviation);
            Plot("BollingerBands", "middleband", _bollingerbands.MiddleBand);
            Plot("BollingerBands", "upperband", _bollingerbands.UpperBand);
            Plot("BollingerBands", "lowerband", _bollingerbands.LowerBand);
            Plot("BollingerBands", "bandwidth", _bollingerbands.BandWidth);
            Plot("BollingerBands", "percentb", _bollingerbands.PercentB);
            Plot("BollingerBands", "price", _bollingerbands.Price);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `BollingerBands` indicator properties:

Indicator History

To get the historical data of the `BollingerBands` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the

`IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class BollingerBandsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private BollingerBands _bb;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _bb = BB(_symbol, 30, 2m);

        var indicatorHistory = IndicatorHistory(_bb, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_bb, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_bb, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var standardDeviation = indicatorHistory.Select(x => ((dynamic)x).StandardDeviation).ToList();
        var middleBand = indicatorHistory.Select(x => ((dynamic)x).MiddleBand).ToList();
        var upperBand = indicatorHistory.Select(x => ((dynamic)x).UpperBand).ToList();
        var lowerBand = indicatorHistory.Select(x => ((dynamic)x).LowerBand).ToList();
        var bandWidth = indicatorHistory.Select(x => ((dynamic)x).BandWidth).ToList();
        var percentB = indicatorHistory.Select(x => ((dynamic)x).PercentB).ToList();
        var price = indicatorHistory.Select(x => ((dynamic)x).Price).ToList();
    }
}
```

Supported Indicators

Chaikin Money Flow

Introduction

The Chaikin Money Flow Index (CMF) is a volume-weighted average of accumulation and distribution over a specified period. CMF = n-day Sum of [(((C - L) - (H - C)) / (H - L)) x Vol] / n-day Sum of Vol Where: n = number of periods, typically 21 H = high L = low C = close Vol = volume https://www.fidelity.com/learning-center/trading-investing/technical-analysis/technical-indicator-guide/cmf

To view the implementation of this indicator, see the LEAN GitHub repository .

Using CMF Indicator

To create an automatic indicator for `ChaikinMoneyFlow` , call the `CMF` helper method from the `QCAlgorithm` class. The `CMF` method creates a `ChaikinMoneyFlow` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class ChaikinMoneyFlowAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ChaikinMoneyFlow _cmf;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _cmf = CMF(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_cmf.IsReady)
        {
            // The current value of _cmf is represented by itself (_cmf)
            // or _cmf.Current.Value
            Plot("ChaikinMoneyFlow", "cmf", _cmf);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `ChaikinMoneyFlow` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ChaikinMoneyFlowAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ChaikinMoneyFlow _chaikinmoneyflow;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _chaikinmoneyflow = new ChaikinMoneyFlow(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _chaikinmoneyflow.Update(bar);

        if (_chaikinmoneyflow.IsReady)
        {
            // The current value of _chaikinmoneyflow is represented by itself (_chaikinmoneyflow)
            // or _chaikinmoneyflow.Current.Value
            Plot("ChaikinMoneyFlow", "chaikinmoneyflow", _chaikinmoneyflow);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `ChaikinMoneyFlow` indicator properties:

Indicator History

To get the historical data of the `ChaikinMoneyFlow` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class ChaikinMoneyFlowAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ChaikinMoneyFlow _cmf;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _cmf = CMF(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_cmf, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_cmf, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_cmf, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Chaikin Oscillator

Introduction

This class is an alias for AccumulationDistributionOscillator (also known as Chaikin Oscillator).

To view the implementation of this indicator, see the LEAN GitHub repository .

Using CO Indicator

To create an automatic indicator for `ChaikinOscillator` , call the `CO` helper method from the `QCAlgorithm` class. The `CO` method creates a `ChaikinOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class ChaikinOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ChaikinOscillator _co;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _co = CO(_symbol, 3, 10);
    }

    public override void OnData(Slice data)
    {

        if (_co.IsReady)
        {
            // The current value of _co is represented by itself (_co)
            // or _co.Current.Value
            Plot("ChaikinOscillator", "co", _co);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `ChaikinOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

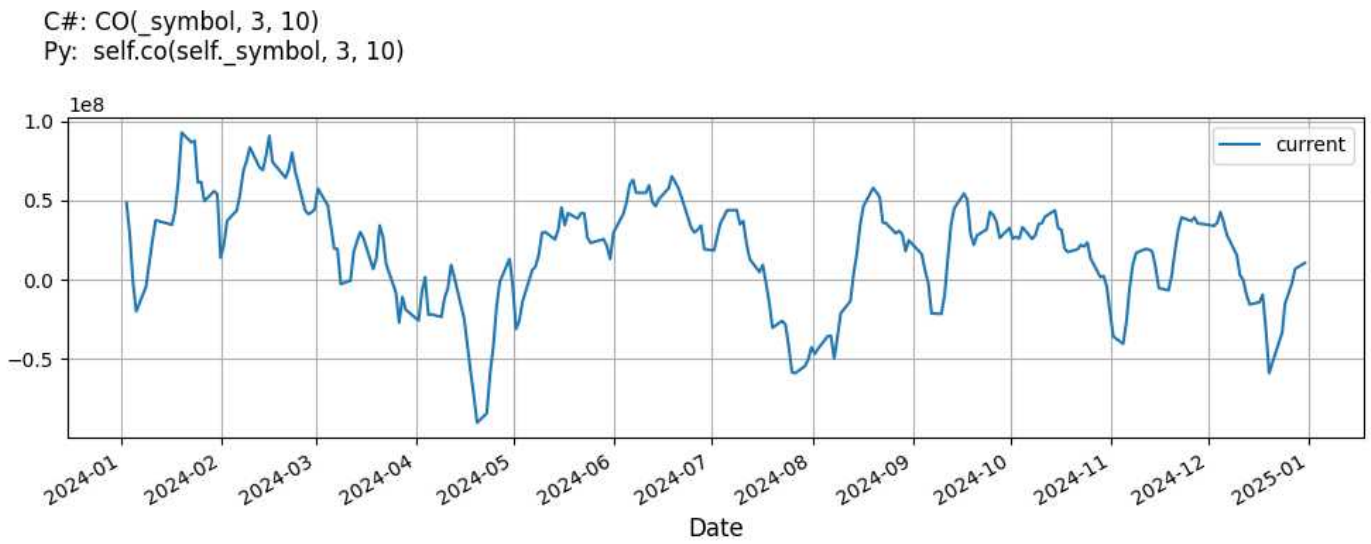
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ChaikinOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ChaikinOscillator _chaikinoscillator;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _chaikinoscillator = new ChaikinOscillator(3, 10);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _chaikinoscillator.Update(bar);

        if (_chaikinoscillator.IsReady)
        {
            // The current value of _chaikinoscillator is represented by itself (_chaikinoscillator)
            // or _chaikinoscillator.Current.Value
            Plot("ChaikinOscillator", "chaikinoscillator", _chaikinoscillator);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `ChaikinOscillator` indicator properties:



Indicator History

To get the historical data of the `ChaikinOscillator` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class ChaikinOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ChaikinOscillator _co;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _co = CO(_symbol, 3, 10);

        var indicatorHistory = IndicatorHistory(_co, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_co, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_co, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Chande Kroll Stop

Introduction

This indicator computes the short stop and lower stop values of the Chande Kroll Stop Indicator. It is used to determine the optimal placement of a stop-loss order.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using CKS Indicator

To create an automatic indicator for `ChandeKrollStop` , call the `CKS` helper method from the `QCAlgorithm` class. The `CKS` method creates a `ChandeKrollStop` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class ChandeKrollStopAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ChandeKrollStop _cks;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _cks = CKS(_symbol, 10, 1, 9);
    }

    public override void OnData(Slice data)
    {

        if (_cks.IsReady)
        {
            // The current value of _cks is represented by itself (_cks)
            // or _cks.Current.Value
            Plot("ChandeKrollStop", "cks", _cks);
            // Plot all properties of abands
            Plot("ChandeKrollStop", "shortstop", _cks.ShortStop);
            Plot("ChandeKrollStop", "longstop", _cks.LongStop);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `ChandeKrollStop` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

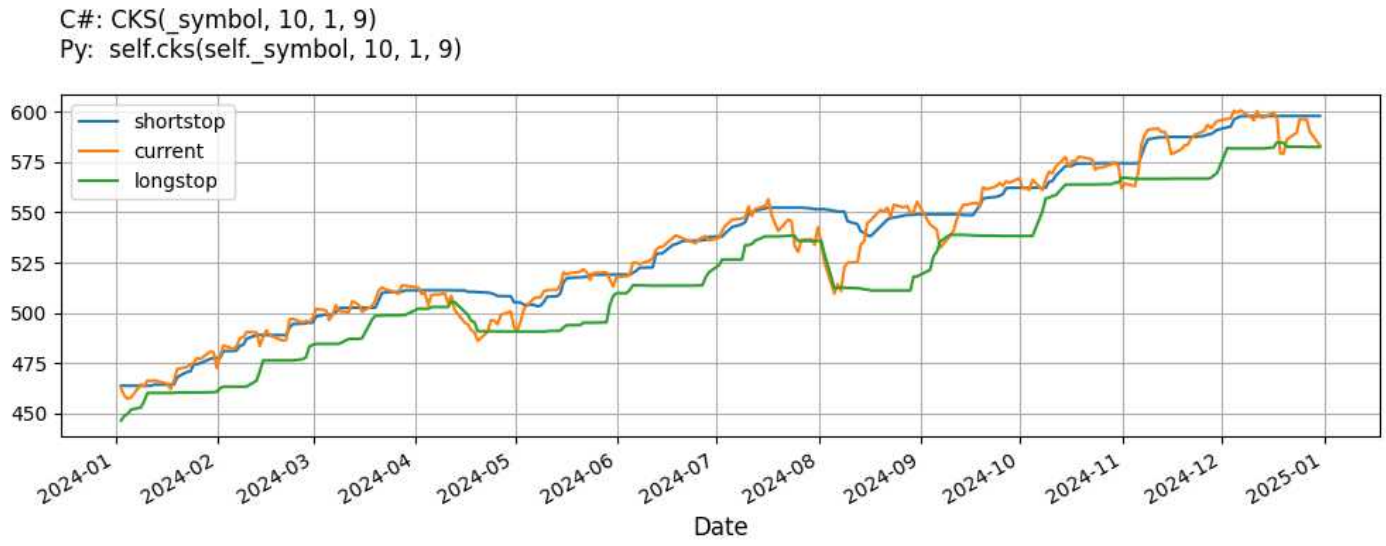
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class ChandeKrollStopAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ChandeKrollStop _chandekrollstop;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _chandekrollstop = new ChandeKrollStop(10, 1, 9);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _chandekrollstop.Update(bar);

        if (_chandekrollstop.IsReady)
        {
            // The current value of _chandekrollstop is represented by itself (_chandekrollstop)
            // or _chandekrollstop.Current.Value
            Plot("ChandeKrollStop", "chandekrollstop", _chandekrollstop);
            // Plot all properties of abands
            Plot("ChandeKrollStop", "shortstop", _chandekrollstop.ShortStop);
            Plot("ChandeKrollStop", "longstop", _chandekrollstop.LongStop);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `ChandeKrollStop` indicator properties:



Indicator History

To get the historical data of the `ChandeKrollStop` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class ChandeKrollStopAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ChandeKrollStop _cks;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _cks = CKS(_symbol, 10, 1, 9);

        var indicatorHistory = IndicatorHistory(_cks, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_cks, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_cks, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var shortStop = indicatorHistory.Select(x => ((dynamic)x).ShortStop).ToList();
        var longStop = indicatorHistory.Select(x => ((dynamic)x).LongStop).ToList();
    }
}
```

Supported Indicators

Chande Momentum Oscillator

Introduction

This indicator computes the Chande Momentum Oscillator (CMO). CMO calculation is mostly identical to RSI. The only difference is in the last step of calculation: RSI = gain / (gain+loss) CMO = (gain-loss) / (gain+loss)

To view the implementation of this indicator, see the LEAN GitHub repository .

Using CMO Indicator

To create an automatic indicator for `ChandeMomentumOscillator` , call the `CMO` helper method from the `QCAlgorithm` class. The `CMO` method creates a `ChandeMomentumOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class ChandeMomentumOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ChandeMomentumOscillator _cmo;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _cmo = CMO(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_cmo.IsReady)
        {
            // The current value of _cmo is represented by itself (_cmo)
            // or _cmo.Current.Value
            Plot("ChandeMomentumOscillator", "cmo", _cmo);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `ChandeMomentumOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

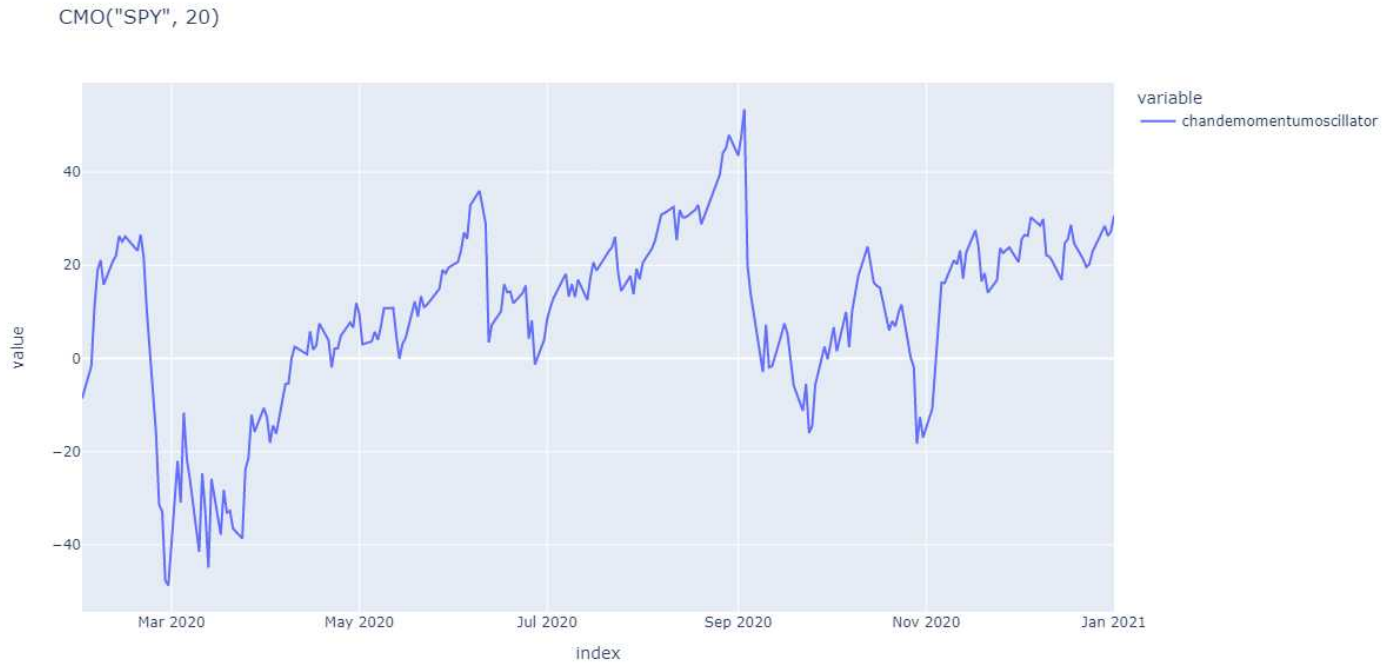
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class ChandeMomentumOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ChandeMomentumOscillator _chandemomentumoscillator;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _chandemomentumoscillator = new ChandeMomentumOscillator(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _chandemomentumoscillator.Update(bar.EndTime, bar.Close);

        if (_chandemomentumoscillator.IsReady)
        {
            // The current value of _chandemomentumoscillator is represented by itself (_chandemomentumoscillator)
            // or _chandemomentumoscillator.Current.Value
            Plot("ChandeMomentumOscillator", "chandemomentumoscillator", _chandemomentumoscillator);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `ChandeMomentumOscillator` indicator properties:



Indicator History

To get the historical data of the `ChandeMomentumOscillator` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class ChandeMomentumOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ChandeMomentumOscillator _cmo;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _cmo = CMO(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_cmo, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_cmo, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_cmo, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Choppiness Index

Introduction

The ChoppinessIndex indicator is an indicator designed to determine if the market is choppy (trading sideways) or not choppy (trading within a trend in either direction)

To view the implementation of this indicator, see the LEAN GitHub repository .

Using CHOP Indicator

To create an automatic indicator for `ChoppinessIndex` , call the `CHOP` helper method from the `QCAlgorithm` class. The `CHOP` method creates a `ChoppinessIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class ChoppinessIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ChoppinessIndex _chop;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _chop = CHOP(_symbol, 14);
    }

    public override void OnData(Slice data)
    {

        if (_chop.IsReady)
        {
            // The current value of _chop is represented by itself (_chop)
            // or _chop.Current.Value
            Plot("ChoppinessIndex", "chop", _chop);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `ChoppinessIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

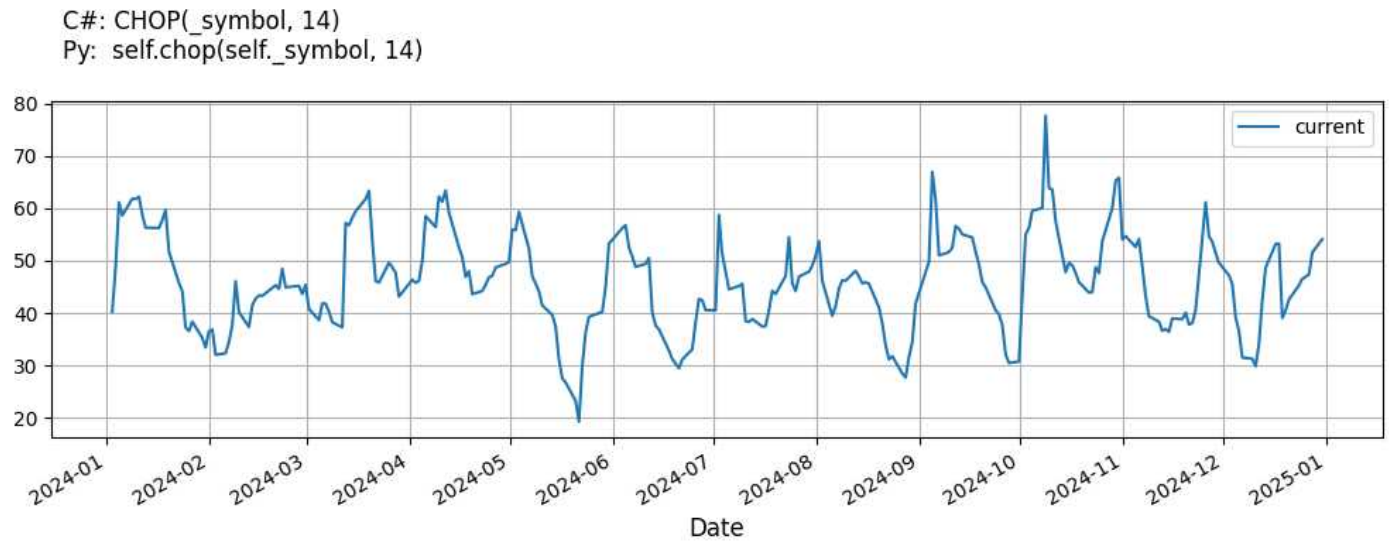
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ChoppinessIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ChoppinessIndex _choppinessindex;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _choppinessindex = new ChoppinessIndex(14);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _choppinessindex.Update(bar);

        if (_choppinessindex.IsReady)
        {
            // The current value of _choppinessindex is represented by itself (_choppinessindex)
            // or _choppinessindex.Current.Value
            Plot("ChoppinessIndex", "choppinessindex", _choppinessindex);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `ChoppinessIndex` indicator properties:



Indicator History

To get the historical data of the `ChoppinessIndex` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class ChoppinessIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ChoppinessIndex _chop;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _chop = CHOP(_symbol, 14);

        var indicatorHistory = IndicatorHistory(_chop, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_chop, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_chop, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Commodity Channel Index

Introduction

This indicator represents the traditional commodity channel index (CCI) CCI = (Typical Price - 20-period SMA of TP) / (.015 * Mean Deviation) Typical Price (TP) = (High + Low + Close)/3 Constant = 0.015 There are four steps to calculating the Mean Deviation, first, subtract the most recent 20-period average of the typical price from each period's typical price. Second, take the absolute values of these numbers. Third, sum the absolute values. Fourth, divide by the total number of periods (20).

To view the implementation of this indicator, see the LEAN GitHub repository .

Using CCI Indicator

To create an automatic indicator for `CommodityChannelIndex` , call the `CCI` helper method from the `QCAlgorithm` class. The `CCI` method creates a `CommodityChannelIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class CommodityChannelIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private CommodityChannelIndex _cci;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _cci = CCI(_symbol, 20, MovingAverageType.Simple);
    }

    public override void OnData(Slice data)
    {

        if (_cci.IsReady)
        {
            // The current value of _cci is represented by itself (_cci)
            // or _cci.Current.Value
            Plot("CommodityChannelIndex", "cci", _cci);
            // Plot all properties of abands
            Plot("CommodityChannelIndex", "typicalpriceaverage", _cci.TypicalPriceAverage);
            Plot("CommodityChannelIndex", "typicalpricemeandeviation", _cci.TypicalPriceMeanDeviation);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `CommodityChannelIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

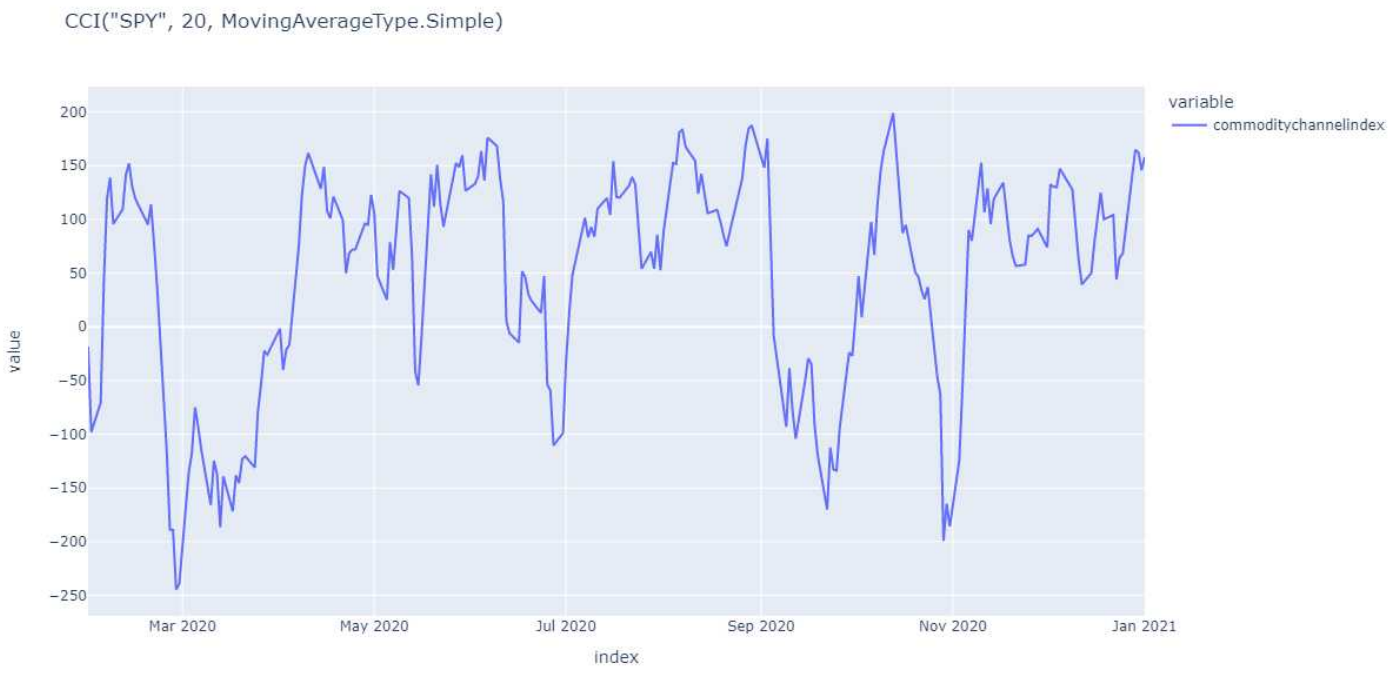
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class CommodityChannelIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private CommodityChannelIndex _commoditychannelindex;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _commoditychannelindex = new CommodityChannelIndex(20, MovingAverageType.Simple);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _commoditychannelindex.Update(bar);

        if (_commoditychannelindex.IsReady)
        {
            // The current value of _commoditychannelindex is represented by itself (_commoditychannelindex)
            // or _commoditychannelindex.Current.Value
            Plot("CommodityChannelIndex", "commoditychannelindex", _commoditychannelindex);
            // Plot all properties of abands
            Plot("CommodityChannelIndex", "typicalpriceaverage", _commoditychannelindex.TypicalPriceAverage);
            Plot("CommodityChannelIndex", "typicalpricemeandeviation",
_commoditychannelindex.TypicalPriceMeanDeviation);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `CommodityChannelIndex` indicator properties:



Indicator History

To get the historical data of the `CommodityChannelIndex` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class CommodityChannelIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private CommodityChannelIndex _cci;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _cci = CCI(_symbol, 20, MovingAverageType.Simple);

        var indicatorHistory = IndicatorHistory(_cci, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_cci, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_cci, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var typicalPriceAverage = indicatorHistory.Select(x => ((dynamic)x).TypicalPriceAverage).ToList();
        var typicalPriceMeanDeviation = indicatorHistory.Select(x =>
((dynamic)x).TypicalPriceMeanDeviation).ToList();
    }
}
```

Supported Indicators

Connors Relative Strength Index

Introduction

This indicator represents the Connors Relative Strength Index (CRSI), a combination of the traditional Relative Strength Index (RSI), a Streak RSI (SRSI), and Percent Rank. This index is designed to provide a more robust measure of market strength by combining momentum, streak behavior, and price change.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using CRSI Indicator

To create an automatic indicator for `ConnorsRelativeStrengthIndex` , call the `CRSI` helper method from the `QCAlgorithm` class. The `CRSI` method creates a `ConnorsRelativeStrengthIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class ConnorsRelativeStrengthIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ConnorsRelativeStrengthIndex _crsi;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _crsi = CRSI(_symbol, 3, 2, 100);
    }

    public override void OnData(Slice data)
    {

        if (_crsi.IsReady)
        {
            // The current value of _crsi is represented by itself (_crsi)
            // or _crsi.Current.Value
            Plot("ConnorsRelativeStrengthIndex", "crsi", _crsi);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `ConnorsRelativeStrengthIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

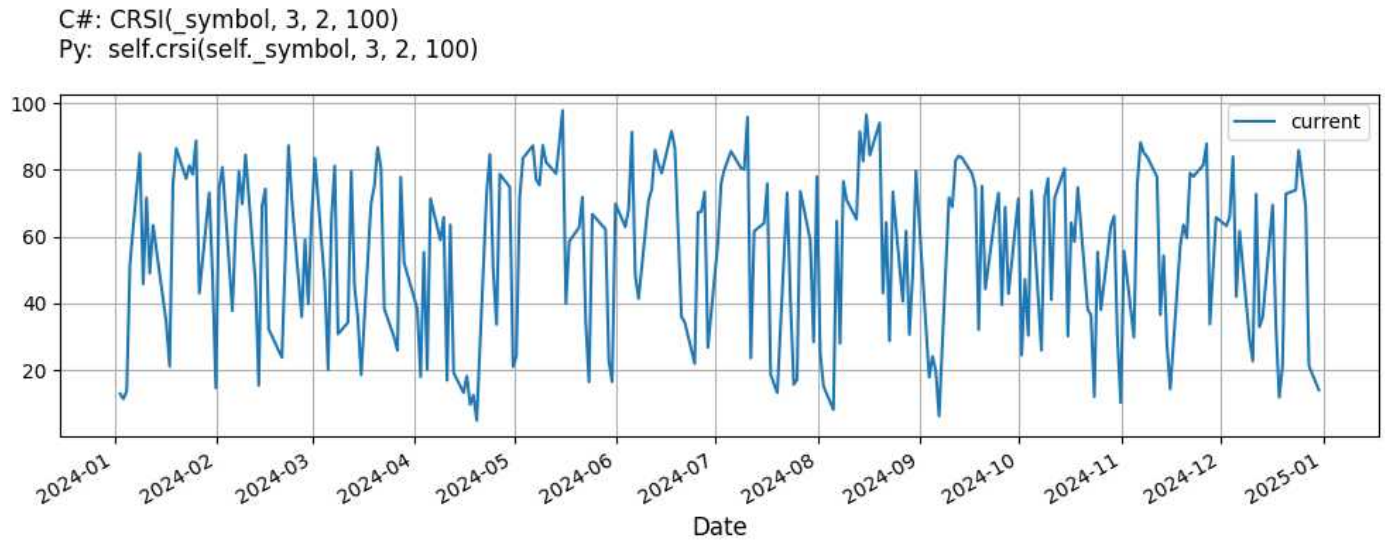
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ConnorsRelativeStrengthIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ConnorsRelativeStrengthIndex _connorsrelativestrengthindex;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _connorsrelativestrengthindex = new ConnorsRelativeStrengthIndex(3, 2, 100);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _connorsrelativestrengthindex.Update(bar.EndTime, bar.Close);

        if (_connorsrelativestrengthindex.IsReady)
        {
            // The current value of _connorsrelativestrengthindex is represented by itself
(_connorsrelativestrengthindex)
            // or _connorsrelativestrengthindex.Current.Value
            Plot("ConnorsRelativeStrengthIndex", "connorsrelativestrengthindex", _connorsrelativestrengthindex);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `ConnorsRelativeStrengthIndex` indicator properties:



Indicator History

To get the historical data of the `ConnorsRelativeStrengthIndex` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class ConnorsRelativeStrengthIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ConnorsRelativeStrengthIndex _crsi;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _crsi = CRSI(_symbol, 3, 2, 100);

        var indicatorHistory = IndicatorHistory(_crsi, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_crsi, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_crsi, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Coppock Curve

Introduction

A momentum indicator developed by Edwin “Sedge” Coppock in October 1965. The goal of this indicator is to identify long-term buying opportunities in the S&P500 and Dow Industrials. source

To view the implementation of this indicator, see the LEAN GitHub repository .

Using CC Indicator

To create an automatic indicator for `CoppockCurve` , call the `CC` helper method from the `QCAlgorithm` class. The `CC` method creates a `CoppockCurve` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class CoppockCurveAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private CoppockCurve _cc;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _cc = CC(_symbol, 11, 14, 10);
    }

    public override void OnData(Slice data)
    {

        if (_cc.IsReady)
        {
            // The current value of _cc is represented by itself (_cc)
            // or _cc.Current.Value
            Plot("CoppockCurve", "cc", _cc);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `CoppockCurve` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

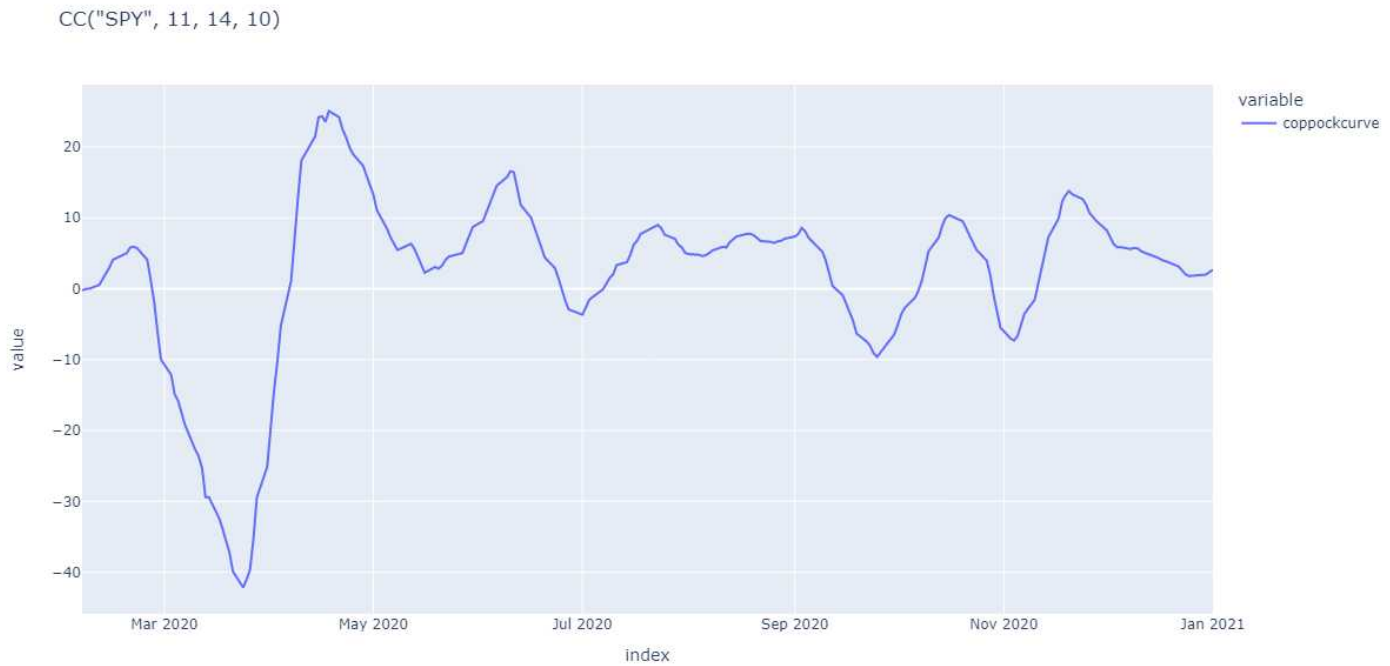
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class CoppockCurveAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private CoppockCurve _coppockcurve;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _coppockcurve = new CoppockCurve(11, 14, 10);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _coppockcurve.Update(bar.EndTime, bar.Close);

        if (_coppockcurve.IsReady)
        {
            // The current value of _coppockcurve is represented by itself (_coppockcurve)
            // or _coppockcurve.Current.Value
            Plot("CoppockCurve", "coppockcurve", _coppockcurve);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `CoppockCurve` indicator properties:



Indicator History

To get the historical data of the `CoppockCurve` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class CoppockCurveAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private CoppockCurve _cc;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _cc = CC(_symbol, 11, 14, 10);

        var indicatorHistory = IndicatorHistory(_cc, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_cc, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_cc, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Correlation

Introduction

The Correlation Indicator is a valuable tool in technical analysis, designed to quantify the degree of relationship between the price movements of a target security (e.g., a stock or ETF) and a reference market index. It measures how closely the target's price changes are aligned with the fluctuations of the index over a specific period of time, providing insights into the target's susceptibility to market movements. A positive correlation indicates that the target tends to move in the same direction as the market index, while a negative correlation suggests an inverse relationship. A correlation close to 0 implies a weak or no linear relationship. Commonly, the SPX index is employed as the benchmark for the overall market when calculating correlation, ensuring a consistent and reliable reference point. This helps traders and investors make informed decisions regarding the risk and behavior of the target security in relation to market trends. The indicator only updates when both assets have a price for a time step. When a bar is missing for one of the assets, the indicator value fills forward to improve the accuracy of the indicator.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using C Indicator

To create an automatic indicator for `Correlation` , call the `C` helper method from the `QCAlgorithm` class. The `C` method creates a `Correlation` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class CorrelationAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private Correlation _c;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _c = C(_symbol, _reference, 20, CorrelationType.Pearson);
    }

    public override void OnData(Slice data)
    {

        if (_c.IsReady)
        {
            // The current value of _c is represented by itself (_c)
            // or _c.Current.Value
            Plot("Correlation", "c", _c);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `Correlation` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

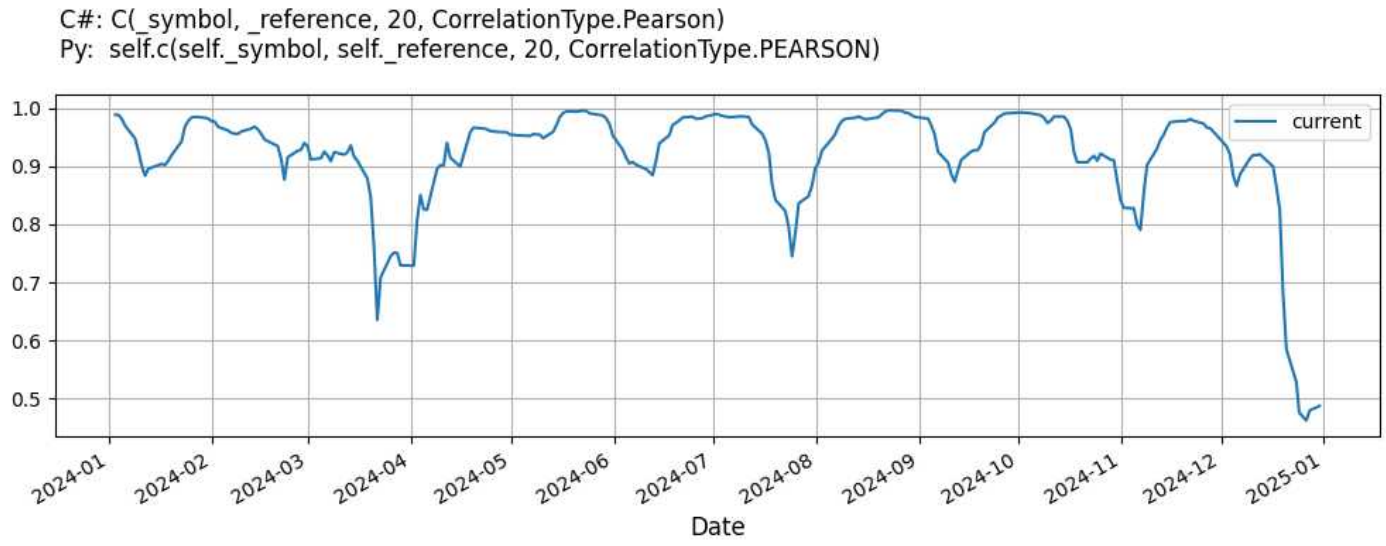
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class CorrelationAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private Correlation _correlation;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _correlation = new Correlation(_symbol, _reference, 20, CorrelationType.Pearson);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _correlation.Update(bar);
        if (data.Bars.TryGetValue(_reference, out bar))
            _correlation.Update(bar);

        if (_correlation.IsReady)
        {
            // The current value of _correlation is represented by itself (_correlation)
            // or _correlation.Current.Value
            Plot("Correlation", "correlation", _correlation);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `Correlation` indicator properties:



Indicator History

To get the historical data of the `Correlation` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class CorrelationAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private Correlation _c;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _c = C(_symbol, _reference, 20, CorrelationType.Pearson);

        var indicatorHistory = IndicatorHistory(_c, new[] { _symbol, _reference }, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_c, new[] { _symbol, _reference }, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_c, new[] { _symbol, _reference }, new DateTime(2024, 7,
1), new DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Covariance

Introduction

This indicator computes the Covariance of two assets using the given Look-Back period. The Covariance of two assets is a measure of their co-movement.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using COV Indicator

To create an automatic indicator for `Covariance` , call the `COV` helper method from the `QCAlgorithm` class. The `COV` method creates a `Covariance` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class CovarianceAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private Covariance _cov;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _cov = COV([_symbol, _reference]);
    }

    public override void OnData(Slice data)
    {

        if (_cov.IsReady)
        {
            // The current value of _cov is represented by itself (_cov)
            // or _cov.Current.Value
            Plot("Covariance", "cov", _cov);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `Covariance` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

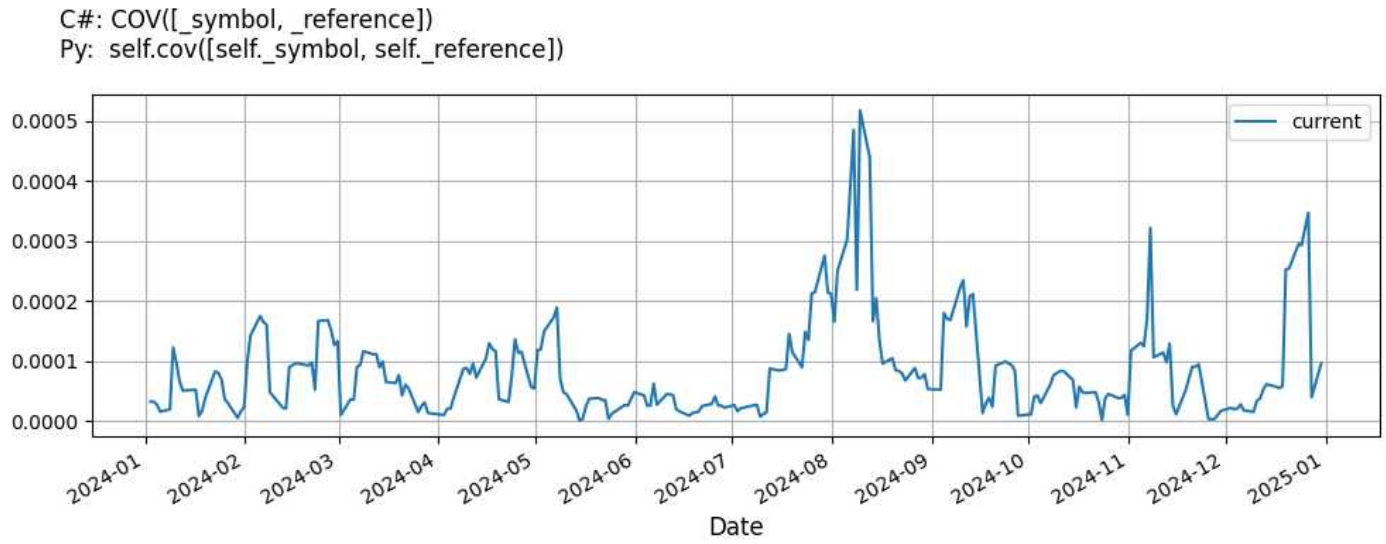
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class CovarianceAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private Covariance _covariance;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _covariance = new Covariance(_symbol, _reference, 5);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _covariance.Update(bar);
        if (data.Bars.TryGetValue(_reference, out bar))
            _covariance.Update(bar);

        if (_covariance.IsReady)
        {
            // The current value of _covariance is represented by itself (_covariance)
            // or _covariance.Current.Value
            Plot("Covariance", "covariance", _covariance);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `Covariance` indicator properties:



Indicator History

To get the historical data of the `Covariance` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class CovarianceAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private Covariance _cov;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _cov = COV([_symbol, _reference]);

        var indicatorHistory = IndicatorHistory(_cov, new[] { _symbol, _reference }, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_cov, new[] { _symbol, _reference }, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_cov, new[] { _symbol, _reference }, new DateTime(2024, 7,
1), new DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

DeMarker Indicator

Introduction

In the DeMarker strategy, for some period of size N, set: DeMax = High - Previous High, and DeMin = Previous Low - Low where, in the prior, if either term is less than zero (DeMax or DeMin), set it to zero. We can now define the indicator itself, DEM, as: DEM = MA(DeMax)/(MA(DeMax)+MA(DeMin)) where MA denotes a Moving Average of period N. https://www.investopedia.com/terms/d/demarkerindicator.asp

To view the implementation of this indicator, see the LEAN GitHub repository .

Using DEM Indicator

To create an automatic indicator for `DeMarkerIndicator` , call the `DEM` helper method from the `QCAlgorithm` class. The `DEM` method creates a `DeMarkerIndicator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class DeMarkerIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DeMarkerIndicator _dem;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _dem = DEM(_symbol, 20, MovingAverageType.Simple);
    }

    public override void OnData(Slice data)
    {

        if (_dem.IsReady)
        {
            // The current value of _dem is represented by itself (_dem)
            // or _dem.Current.Value
            Plot("DeMarkerIndicator", "dem", _dem);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `DeMarkerIndicator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

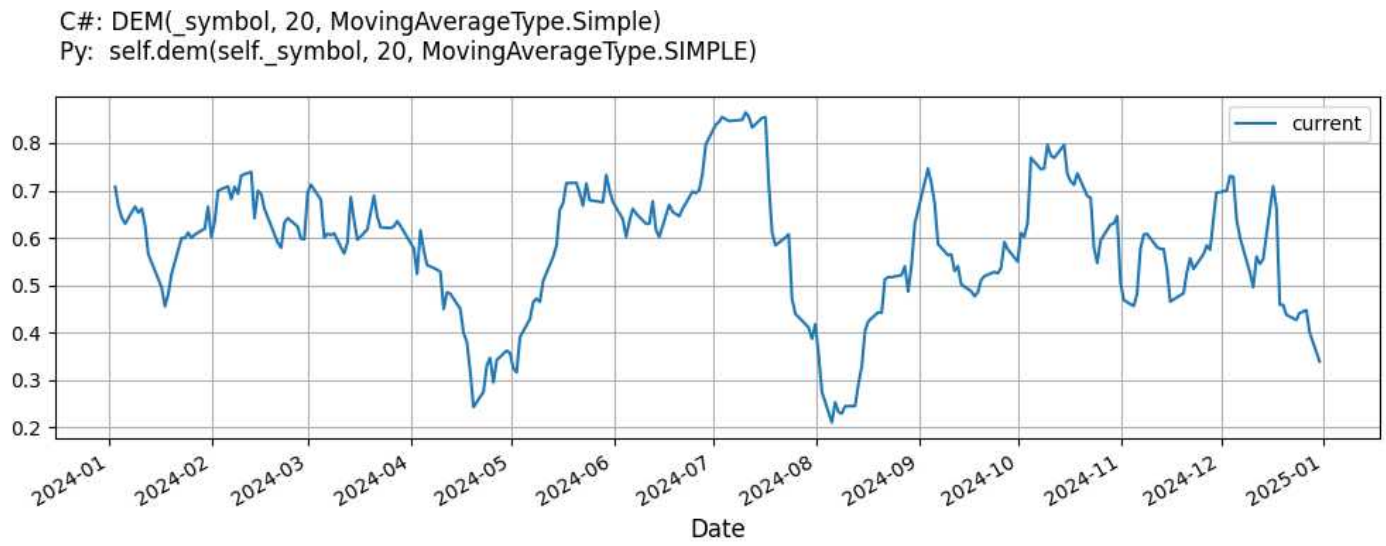
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class DeMarkerIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DeMarkerIndicator _demarkerindicator;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _demarkerindicator = new DeMarkerIndicator(20, MovingAverageType.Simple);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _demarkerindicator.Update(bar);

        if (_demarkerindicator.IsReady)
        {
            // The current value of _demarkerindicator is represented by itself (_demarkerindicator)
            // or _demarkerindicator.Current.Value
            Plot("DeMarkerIndicator", "demarkerindicator", _demarkerindicator);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `DeMarkerIndicator` indicator properties:



Indicator History

To get the historical data of the `DeMarkerIndicator` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class DeMarkerIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DeMarkerIndicator _dem;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _dem = DEM(_symbol, 20, MovingAverageType.Simple);

        var indicatorHistory = IndicatorHistory(_dem, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_dem, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_dem, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Delta

Introduction

Option Delta indicator that calculate the delta of an option

To view the implementation of this indicator, see the LEAN GitHub repository .

Using D Indicator

To create an automatic indicator for `Delta` , call the `D` helper method from the `QCAlgorithm` class. The `D` method creates a `Delta` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class DeltaAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _option, _mirrorOption;
    private Delta _d;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 20);

        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _option = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920C00564000");
        _mirrorOption = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920P00564000");
        AddOptionContract(_option, Resolution.Daily);
        AddOptionContract(_mirrorOption, Resolution.Daily);

        _d = D(_option, _mirrorOption);
    }

    public override void OnData(Slice data)
    {
        if (_d.IsReady)
        {
            // The current value of _d is represented by itself (_d)
            // or _d.Current.Value
            Plot("Delta", "d", _d);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `Delta` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

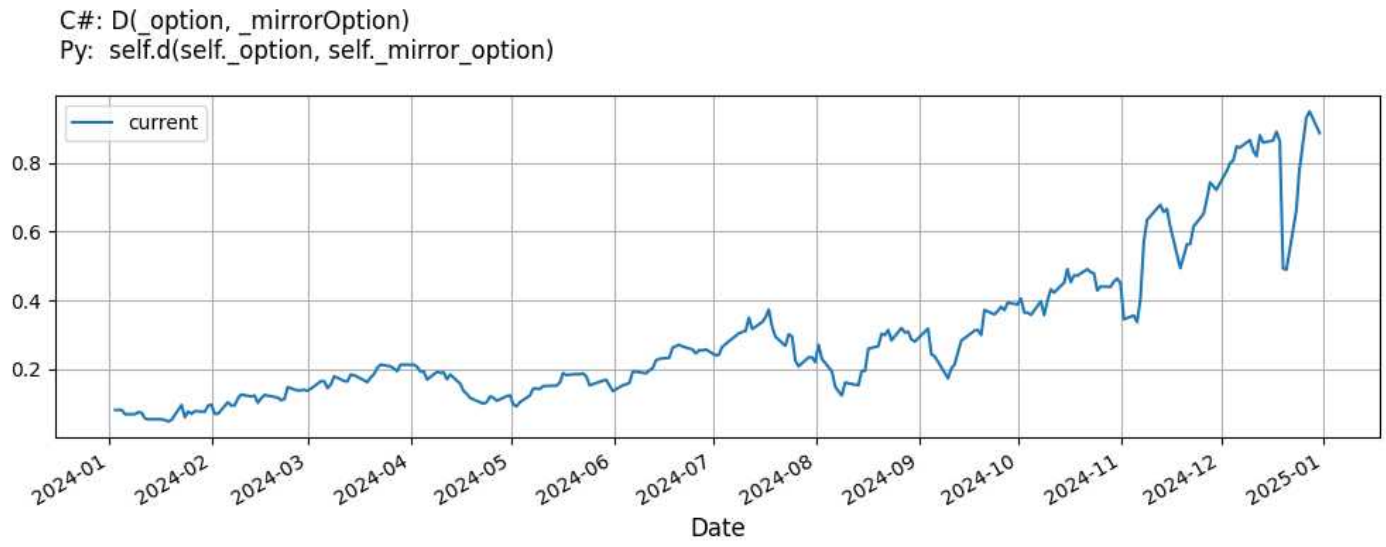
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class DeltaAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _option, _mirrorOption;
    private Delta _delta;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 20);

        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        var interestRateModel = new InterestRateProvider();
        var dividendYieldModel = new DividendYieldProvider(_symbol);

        _option = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920C00564000");
        _mirrorOption = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920P00564000");
        AddOptionContract(_option, Resolution.Daily);
        AddOptionContract(_mirrorOption, Resolution.Daily);

        _delta = new Delta(_option, interestRateModel, dividendYieldModel, _mirrorOption);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _delta.Update(new IndicatorDataPoint(_symbol, bar.EndTime, bar.Close));
        if (data.QuoteBars.TryGetValue(_option, out var quoteBar))
            _delta.Update(new IndicatorDataPoint(_option, quoteBar.EndTime, quoteBar.Close));
        if (data.QuoteBars.TryGetValue(_mirrorOption, out quoteBar))
            _delta.Update(new IndicatorDataPoint(_mirrorOption, quoteBar.EndTime, quoteBar.Close));

        if (_delta.IsReady)
        {
            // The current value of _delta is represented by itself (_delta)
            // or _delta.Current.Value
            Plot("Delta", "delta", _delta);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `Delta` indicator properties:



Indicator History

To get the historical data of the `Delta` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory`

method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class DeltaAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _option, _mirrorOption;
    private Delta _d;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 20);

        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _option = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920C00564000");
        _mirrorOption = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920P00564000");
        AddOptionContract(_option, Resolution.Daily);
        AddOptionContract(_mirrorOption, Resolution.Daily);

        _d = D(_option, _mirrorOption);

        var indicatorHistory = IndicatorHistory(_d, new[] { _symbol, _option, _mirrorOption }, 100,
Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_d, new[] { _symbol, _option, _mirrorOption },
TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_d, new[] { _symbol, _option, _mirrorOption }, new
DateTime(2024, 7, 1), new DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Derivative Oscillator

Introduction

This indicator represents the Derivative Oscillator Indicator, utilizing a moving average convergence-divergence (MACD) histogram to a double-smoothed relative strength index (RSI).

To view the implementation of this indicator, see the LEAN GitHub repository .

Using DO Indicator

To create an automatic indicator for `DerivativeOscillator` , call the `DO` helper method from the `QCAlgorithm` class. The `DO` method creates a `DerivativeOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class DerivativeOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DerivativeOscillator _do;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _do = DO(_symbol, 14, 5, 3, 9);
    }

    public override void OnData(Slice data)
    {

        if (_do.IsReady)
        {
            // The current value of _do is represented by itself (_do)
            // or _do.Current.Value
            Plot("DerivativeOscillator", "do", _do);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `DerivativeOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

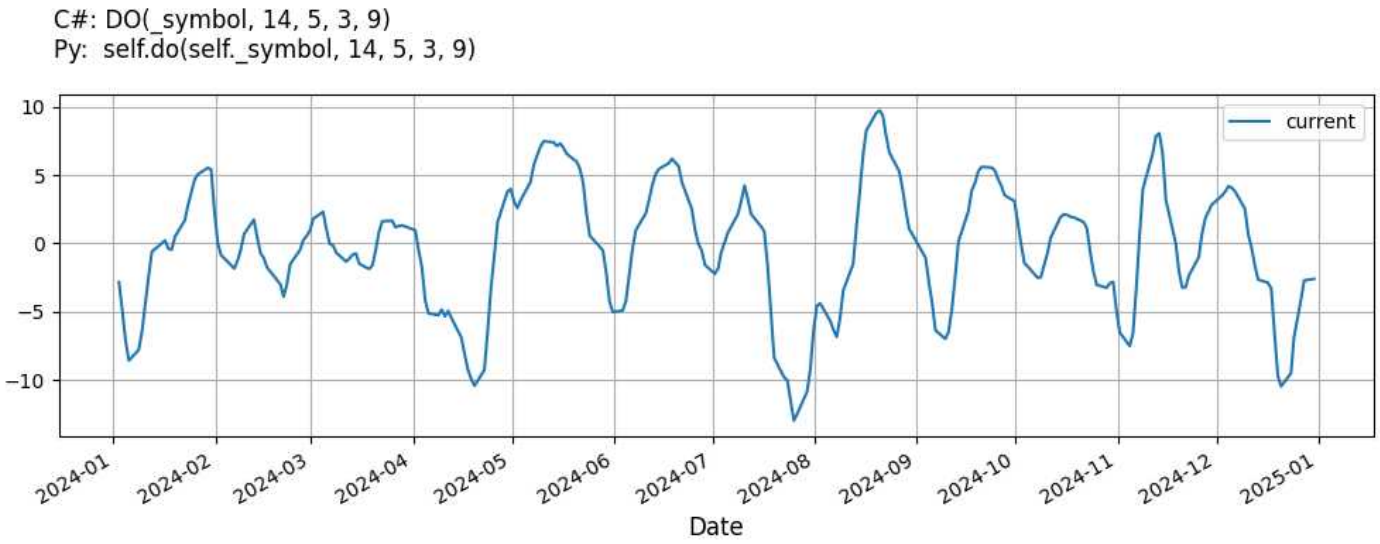
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class DerivativeOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DerivativeOscillator _derivativeoscillator;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _derivativeoscillator = new DerivativeOscillator(14, 5, 3, 9);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _derivativeoscillator.Update(bar.EndTime, bar.Close);

        if (_derivativeoscillator.IsReady)
        {
            // The current value of _derivativeoscillator is represented by itself (_derivativeoscillator)
            // or _derivativeoscillator.Current.Value
            Plot("DerivativeOscillator", "derivativeoscillator", _derivativeoscillator);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `DerivativeOscillator` indicator properties:



Indicator History

To get the historical data of the `DerivativeOscillator` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class DerivativeOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DerivativeOscillator _do;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _do = DO(_symbol, 14, 5, 3, 9);

        var indicatorHistory = IndicatorHistory(_do, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_do, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_do, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Detrended Price Oscillator

Introduction

The Detrended Price Oscillator is an indicator designed to remove trend from price and make it easier to identify cycles. DPO does not extend to the last date because it is based on a displaced moving average. Is estimated as Price {X/2 + 1} periods ago less the X-period simple moving average. E.g.DPO(20) equals price 11 days ago less the 20-day SMA.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using DPO Indicator

To create an automatic indicator for `DetrendedPriceOscillator` , call the `DPO` helper method from the `QCAlgorithm` class. The `DPO` method creates a `DetrendedPriceOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class DetrendedPriceOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DetrendedPriceOscillator _dpo;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _dpo = DPO(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_dpo.IsReady)
        {
            // The current value of _dpo is represented by itself (_dpo)
            // or _dpo.Current.Value
            Plot("DetrendedPriceOscillator", "dpo", _dpo);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `DetrendedPriceOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

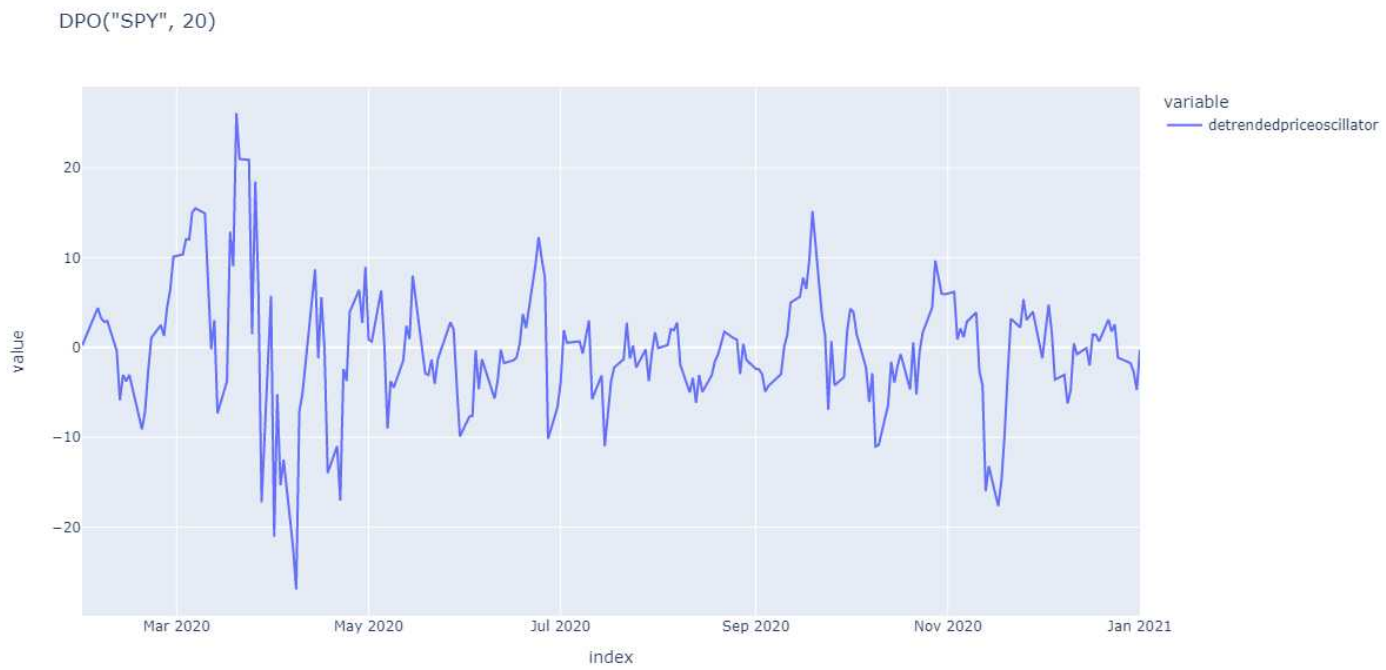
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class DetrendedPriceOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DetrendedPriceOscillator _detrendedpriceoscillator;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _detrendedpriceoscillator = new DetrendedPriceOscillator(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _detrendedpriceoscillator.Update(bar.EndTime, bar.Close);

        if (_detrendedpriceoscillator.IsReady)
        {
            // The current value of _detrendedpriceoscillator is represented by itself (_detrendedpriceoscillator)
            // or _detrendedpriceoscillator.Current.Value
            Plot("DetrendedPriceOscillator", "detrendedpriceoscillator", _detrendedpriceoscillator);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `DetrendedPriceOscillator` indicator properties:



Indicator History

To get the historical data of the `DetrendedPriceOscillator` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class DetrendedPriceOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DetrendedPriceOscillator _dpo;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _dpo = DPO(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_dpo, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_dpo, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_dpo, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Donchian Channel

Introduction

This indicator computes the upper and lower band of the Donchian Channel. The upper band is computed by finding the highest high over the given period. The lower band is computed by finding the lowest low over the given period. The primary output value of the indicator is the mean of the upper and lower band for the given timeframe.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using DCH Indicator

To create an automatic indicator for `DonchianChannel` , call the `DCH` helper method from the `QCAlgorithm` class. The `DCH` method creates a `DonchianChannel` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class DonchianChannelAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DonchianChannel _dch;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _dch = DCH(_symbol, 20, 20);
    }

    public override void OnData(Slice data)
    {

        if (_dch.IsReady)
        {
            // The current value of _dch is represented by itself (_dch)
            // or _dch.Current.Value
            Plot("DonchianChannel", "dch", _dch);
            // Plot all properties of abands
            Plot("DonchianChannel", "upperband", _dch.UpperBand);
            Plot("DonchianChannel", "lowerband", _dch.LowerBand);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `DonchianChannel` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

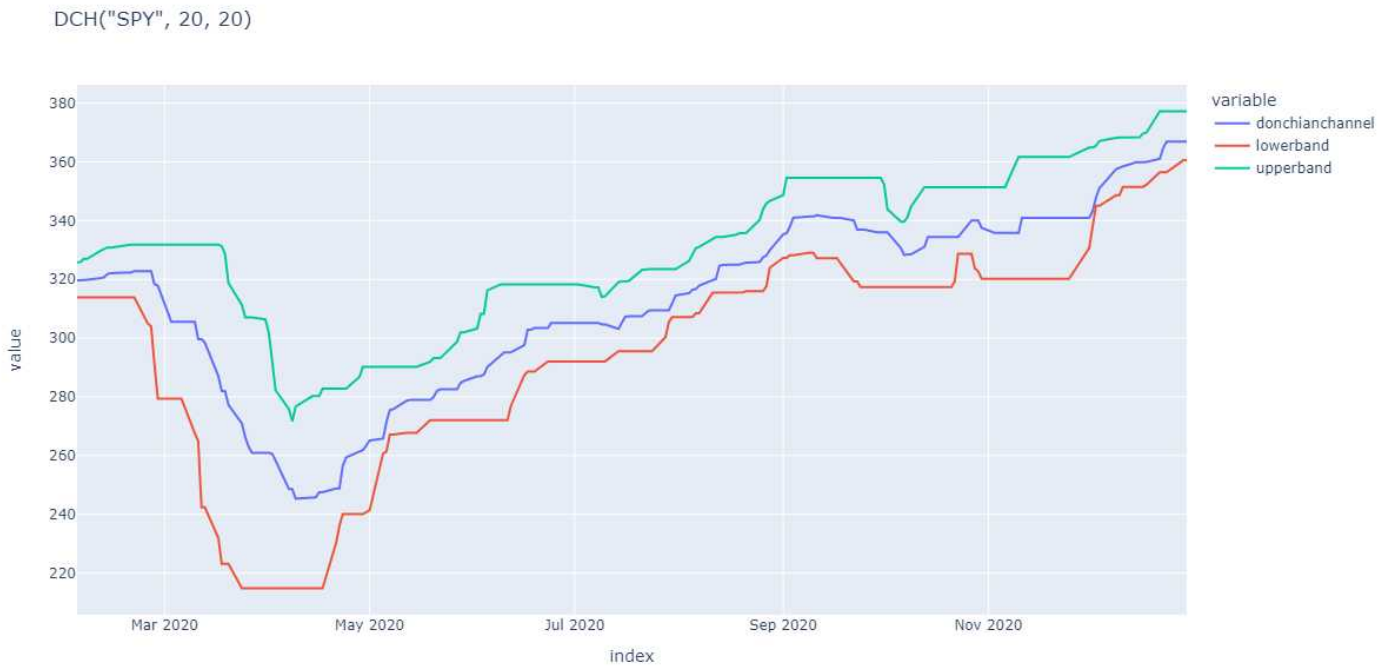
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class DonchianChannelAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DonchianChannel _donchianchannel;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _donchianchannel = new DonchianChannel(20, 20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _donchianchannel.Update(bar);

        if (_donchianchannel.IsReady)
        {
            // The current value of _donchianchannel is represented by itself (_donchianchannel)
            // or _donchianchannel.Current.Value
            Plot("DonchianChannel", "donchianchannel", _donchianchannel);
            // Plot all properties of abands
            Plot("DonchianChannel", "upperband", _donchianchannel.UpperBand);
            Plot("DonchianChannel", "lowerband", _donchianchannel.LowerBand);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `DonchianChannel` indicator properties:



Indicator History

To get the historical data of the `DonchianChannel` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class DonchianChannelAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DonchianChannel _dch;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _dch = DCH(_symbol, 20, 20);

        var indicatorHistory = IndicatorHistory(_dch, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_dch, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_dch, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var upperBand = indicatorHistory.Select(x => ((dynamic)x).UpperBand).ToList();
        var lowerBand = indicatorHistory.Select(x => ((dynamic)x).LowerBand).ToList();
    }
}
```

Supported Indicators

Double Exponential Moving Average

Introduction

This indicator computes the Double Exponential Moving Average (DEMA). The Double Exponential Moving Average is calculated with the following formula: EMA2 = EMA(EMA(t,period),period) DEMA = 2 * EMA(t,period) - EMA2 The Generalized DEMA (GD) is calculated with the following formula: GD = (volumeFactor+1) * EMA(t,period) - volumeFactor * EMA2

To view the implementation of this indicator, see the LEAN GitHub repository .

Using DEMA Indicator

To create an automatic indicator for `DoubleExponentialMovingAverage` , call the `DEMA` helper method from the `QCAlgorithm` class. The `DEMA` method creates a `DoubleExponentialMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

C#

```
public class DoubleExponentialMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DoubleExponentialMovingAverage _dema;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _dema = DEMA(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_dema.IsReady)
        {
            // The current value of _dema is represented by itself (_dema)
            // or _dema.Current.Value
            Plot("DoubleExponentialMovingAverage", "dema", _dema);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `DoubleExponentialMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

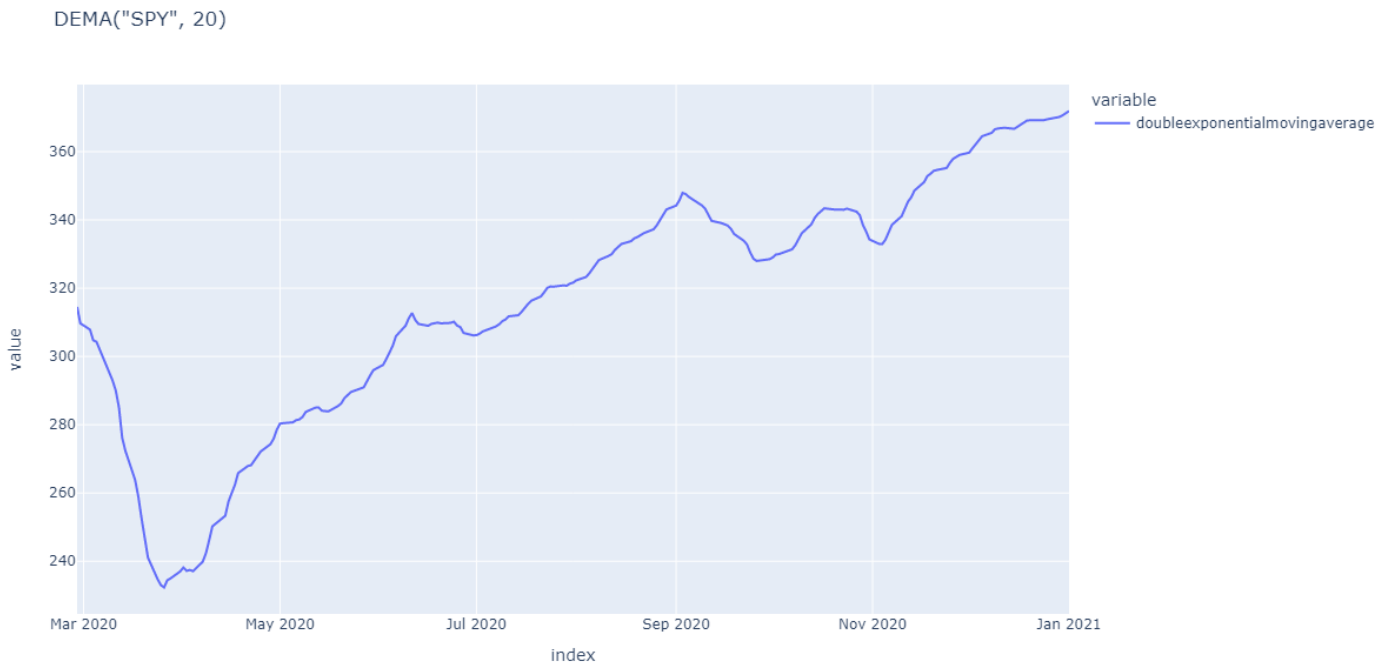
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class DoubleExponentialMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DoubleExponentialMovingAverage _doubleexponentialmovingaverage;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _doubleexponentialmovingaverage = new DoubleExponentialMovingAverage(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _doubleexponentialmovingaverage.Update(bar.EndTime, bar.Close);

        if (_doubleexponentialmovingaverage.IsReady)
        {
            // The current value of _doubleexponentialmovingaverage is represented by itself
(_doubleexponentialmovingaverage)
            // or _doubleexponentialmovingaverage.Current.Value
            Plot("DoubleExponentialMovingAverage", "doubleexponentialmovingaverage",
_doubleexponentialmovingaverage);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `DoubleExponentialMovingAverage` indicator properties:



Indicator History

To get the historical data of the `DoubleExponentialMovingAverage` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class DoubleExponentialMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private DoubleExponentialMovingAverage _dema;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _dema = DEMA(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_dema, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_dema, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_dema, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Ease Of Movement Value

Introduction

This indicator computes the n-period Ease of Movement Value using the following: MID = (high_1 + low_1)/2 - (high_0 + low_0)/2 RATIO = (currentVolume/10000) / (high_1 - low_1) EMV = MID/RATIO _SMA = n-period of EMV Returns _SMA source

To view the implementation of this indicator, see the LEAN GitHub repository .

Using EMV Indicator

To create an automatic indicator for `EaseOfMovementValue` , call the `EMV` helper method from the `QCAlgorithm` class. The `EMV` method creates a `EaseOfMovementValue` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class EaseOfMovementValueAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private EaseOfMovementValue _emv;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _emv = EMV(_symbol, 1, 10000);
    }

    public override void OnData(Slice data)
    {

        if (_emv.IsReady)
        {
            // The current value of _emv is represented by itself (_emv)
            // or _emv.Current.Value
            Plot("EaseOfMovementValue", "emv", _emv);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `EaseOfMovementValue` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

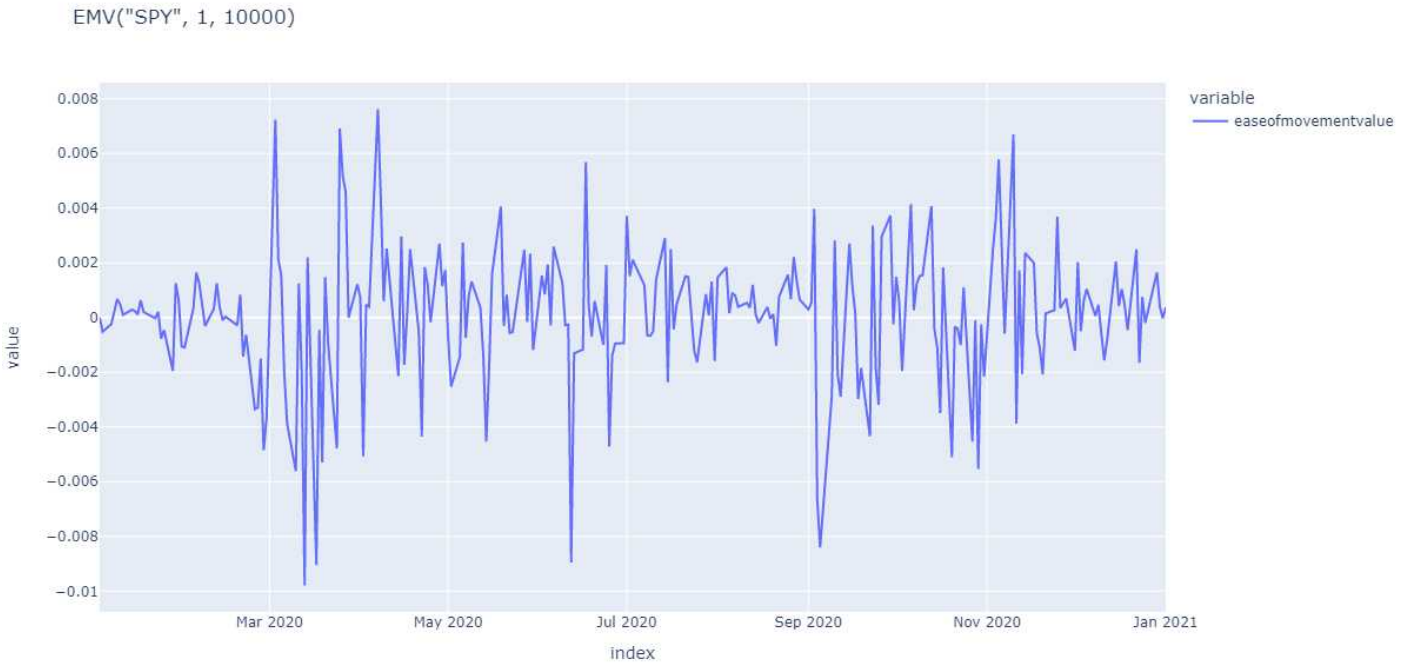
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class EaseOfMovementValueAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private EaseOfMovementValue _easeofmovementvalue;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _easeofmovementvalue = new EaseOfMovementValue(1, 10000);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _easeofmovementvalue.Update(bar);

        if (_easeofmovementvalue.IsReady)
        {
            // The current value of _easeofmovementvalue is represented by itself (_easeofmovementvalue)
            // or _easeofmovementvalue.Current.Value
            Plot("EaseOfMovementValue", "easeofmovementvalue", _easeofmovementvalue);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `EaseOfMovementValue` indicator properties:



Indicator History

To get the historical data of the `EaseOfMovementValue` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class EaseOfMovementValueAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private EaseOfMovementValue _emv;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _emv = EMV(_symbol, 1, 10000);

        var indicatorHistory = IndicatorHistory(_emv, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_emv, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_emv, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Exponential Moving Average

Introduction

This indicator represents the traditional exponential moving average indicator (EMA). When the indicator is ready, the first value of the EMA is equivalent to the simple moving average. After the first EMA value, the EMA value is a function of the previous EMA value. Therefore, depending on the number of samples you feed into the indicator, it can provide different EMA values for a single security and lookback period. To make the indicator values consistent across time, warm up the indicator with all the trailing security price history.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using EMA Indicator

To create an automatic indicator for `ExponentialMovingAverage` , call the `EMA` helper method from the `QCAlgorithm` class. The `EMA` method creates a `ExponentialMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

C#

```
public class ExponentialMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ExponentialMovingAverage _ema;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ema = EMA(_symbol, 20, 0.5m);
    }

    public override void OnData(Slice data)
    {

        if (_ema.IsReady)
        {
            // The current value of _ema is represented by itself (_ema)
            // or _ema.Current.Value
            Plot("ExponentialMovingAverage", "ema", _ema);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `ExponentialMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

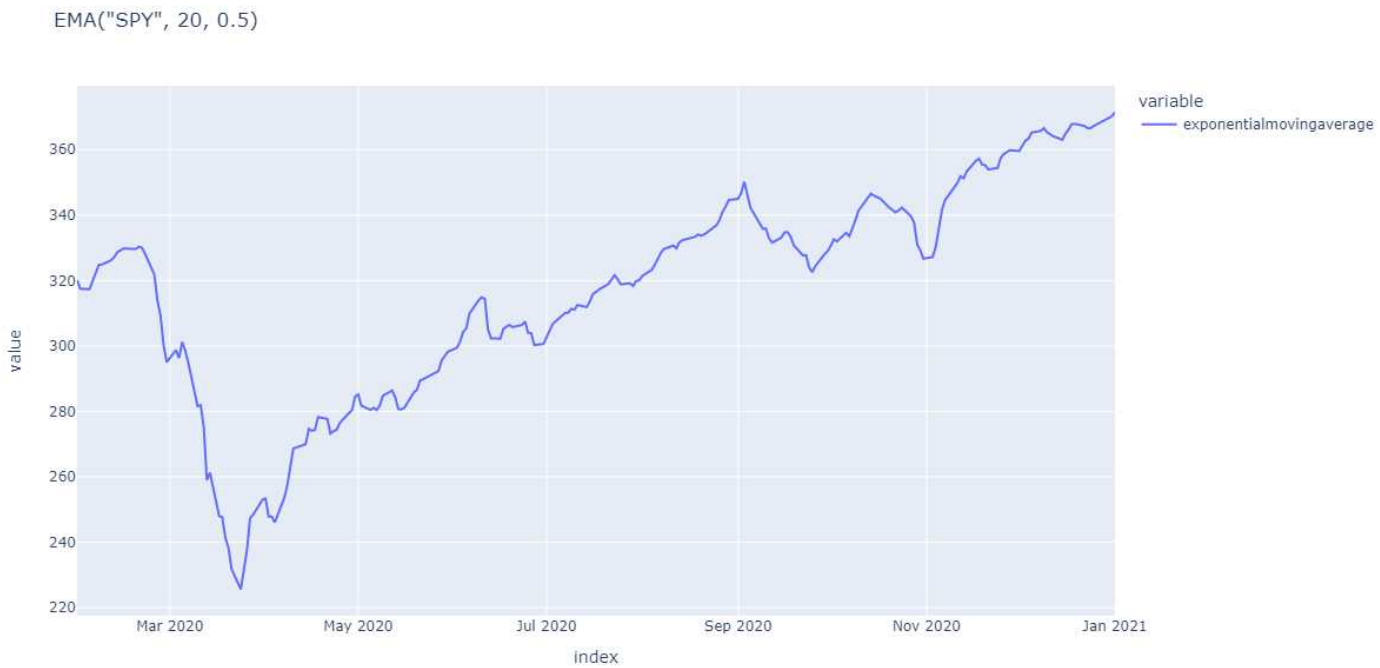
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ExponentialMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ExponentialMovingAverage _exponentialmovingaverage;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _exponentialmovingaverage = new ExponentialMovingAverage(20, 0.5m);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _exponentialmovingaverage.Update(bar.EndTime, bar.Close);

        if (_exponentialmovingaverage.IsReady)
        {
            // The current value of _exponentialmovingaverage is represented by itself (_exponentialmovingaverage)
            // or _exponentialmovingaverage.Current.Value
            Plot("ExponentialMovingAverage", "exponentialmovingaverage", _exponentialmovingaverage);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `ExponentialMovingAverage` indicator properties:



Indicator History

To get the historical data of the `ExponentialMovingAverage` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class ExponentialMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ExponentialMovingAverage _ema;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ema = EMA(_symbol, 20, 0.5m);

        var indicatorHistory = IndicatorHistory(_ema, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_ema, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_ema, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Filtered Identity

Introduction

This indicator represents an indicator that is a ready after ingesting a single sample and always returns the same value as it is given if it passes a filter condition

To view the implementation of this indicator, see the LEAN GitHub repository .

Using FilteredIdentity Indicator

To create an automatic indicator for `FilteredIdentity` , call the `FilteredIdentity` helper method from the `QCAlgorithm` class. The `FilteredIdentity` method creates a `FilteredIdentity` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class FilteredIdentityAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private FilteredIdentity _filteredidentity;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _filteredidentity = FilteredIdentity(_symbol, filter: x =>  (x as TradeBar).Close > (x as TradeBar).Open);
    }

    public override void OnData(Slice data)
    {

        if (_filteredidentity.IsReady)
        {
            // The current value of _filteredidentity is represented by itself (_filteredidentity)
            // or _filteredidentity.Current.Value
            Plot("FilteredIdentity", "filteredidentity", _filteredidentity);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `FilteredIdentity` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class FilteredIdentityAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private FilteredIdentity _filteredidentity;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _filteredidentity = new FilteredIdentity(filter: x =>  (x as TradeBar).Close > (x as TradeBar).Open);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _filteredidentity.Update(bar);

        if (_filteredidentity.IsReady)
        {
            // The current value of _filteredidentity is represented by itself (_filteredidentity)
            // or _filteredidentity.Current.Value
            Plot("FilteredIdentity", "filteredidentity", _filteredidentity);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `FilteredIdentity` indicator properties:

Indicator History

To get the historical data of the `FilteredIdentity` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class FilteredIdentityAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private FilteredIdentity _filteredidentity;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _filteredidentity = FilteredIdentity(_symbol, filter: x =>  (x as TradeBar).Close > (x as TradeBar).Open);

        var indicatorHistory = IndicatorHistory(_filteredidentity, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_filteredidentity, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_filteredidentity, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Fisher Transform

Introduction

The Fisher transform is a mathematical process which is used to convert any data set to a modified data set whose Probability Distribution Function is approximately Gaussian. Once the Fisher transform is computed, the transformed data can then be analyzed in terms of it's deviation from the mean. The equation is y = .5 * ln [1 + x / 1 - x] where x is the input y is the output ln is the natural logarithm The Fisher transform has much sharper turning points than other indicators such as MACD For more info, read chapter 1 of Cybernetic Analysis for Stocks and Futures by John F. Ehlers We are implementing the latest version of this indicator found at Fig. 4 of http://www.mesasoftware.com/papers/UsingTheFisherTransform.pdf

To view the implementation of this indicator, see the LEAN GitHub repository .

Using FISH Indicator

To create an automatic indicator for `FisherTransform` , call the `FISH` helper method from the `QCAlgorithm` class. The `FISH` method creates a `FisherTransform` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class FisherTransformAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private FisherTransform _fish;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _fish = FISH(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_fish.IsReady)
        {
            // The current value of _fish is represented by itself (_fish)
            // or _fish.Current.Value
            Plot("FisherTransform", "fish", _fish);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `FisherTransform` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

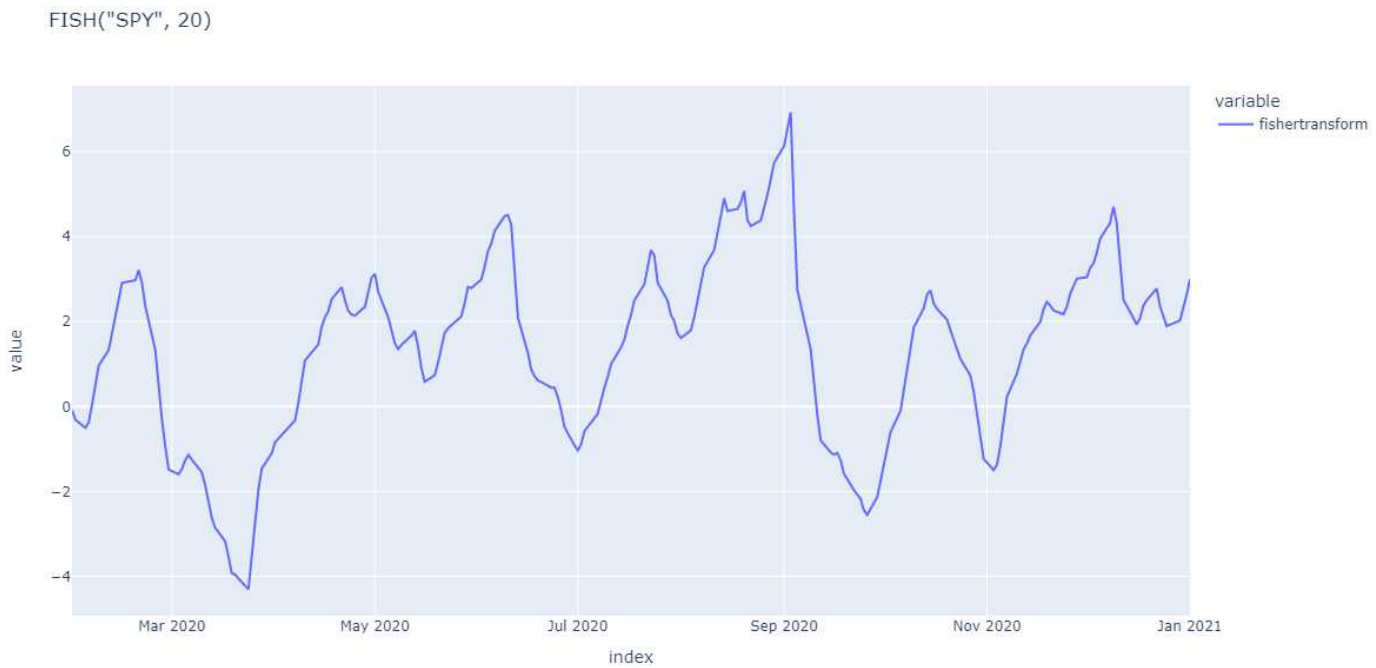
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class FisherTransformAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private FisherTransform _fishertransform;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _fishertransform = new FisherTransform(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _fishertransform.Update(bar);

        if (_fishertransform.IsReady)
        {
            // The current value of _fishertransform is represented by itself (_fishertransform)
            // or _fishertransform.Current.Value
            Plot("FisherTransform", "fishertransform", _fishertransform);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `FisherTransform` indicator properties:



Indicator History

To get the historical data of the `FisherTransform` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class FisherTransformAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private FisherTransform _fish;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _fish = FISH(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_fish, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_fish, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_fish, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Force Index

Introduction

The Force Index is calculated by comparing the current market price with the previous market price and multiplying its difference with the traded volume during a specific time period.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using FI Indicator

To create an automatic indicator for `ForceIndex` , call the `FI` helper method from the `QCAlgorithm` class. The `FI` method creates a `ForceIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class ForceIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ForceIndex _fi;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _fi = FI(_symbol, 13);
    }

    public override void OnData(Slice data)
    {

        if (_fi.IsReady)
        {
            // The current value of _fi is represented by itself (_fi)
            // or _fi.Current.Value
            Plot("ForceIndex", "fi", _fi);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `ForceIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

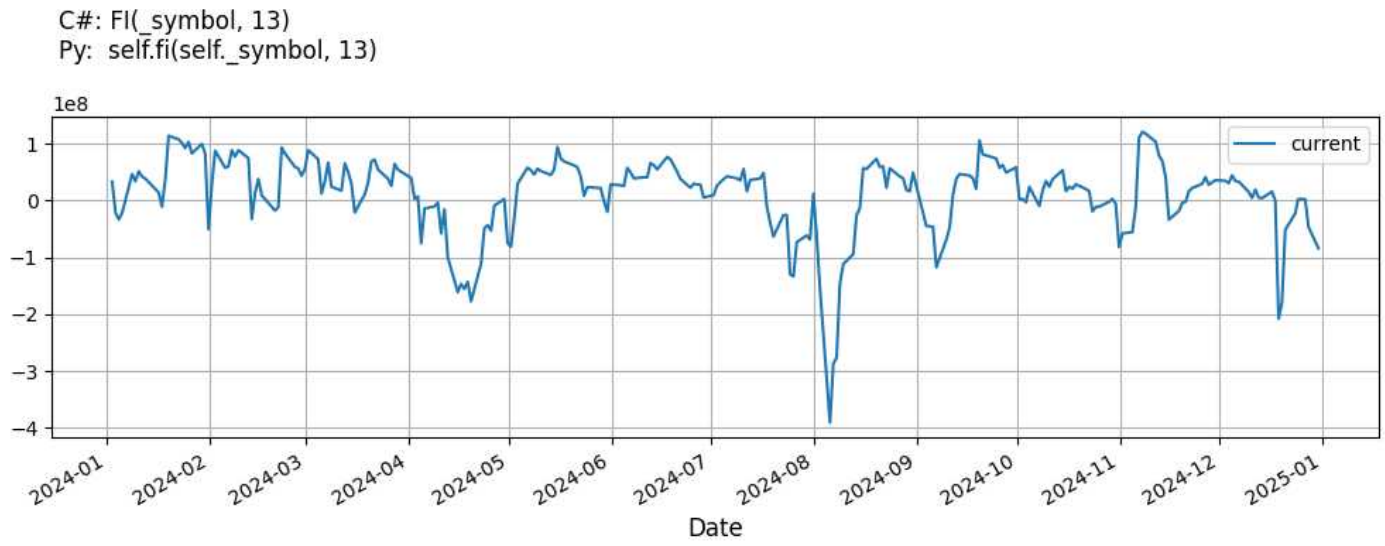
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ForceIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ForceIndex _forceindex;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _forceindex = new ForceIndex(13);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _forceindex.Update(bar);

        if (_forceindex.IsReady)
        {
            // The current value of _forceindex is represented by itself (_forceindex)
            // or _forceindex.Current.Value
            Plot("ForceIndex", "forceindex", _forceindex);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `ForceIndex` indicator properties:



Indicator History

To get the historical data of the `ForceIndex` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class ForceIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ForceIndex _fi;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _fi = FI(_symbol, 13);

        var indicatorHistory = IndicatorHistory(_fi, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_fi, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_fi, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Fractal Adaptive Moving Average

Introduction

The Fractal Adaptive Moving Average (FRAMA) by John Ehlers

To view the implementation of this indicator, see the LEAN GitHub repository .

Using FRAMA Indicator

To create an automatic indicator for `FractalAdaptiveMovingAverage` , call the `FRAMA` helper method from the `QCAlgorithm` class. The `FRAMA` method creates a `FractalAdaptiveMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class FractalAdaptiveMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private FractalAdaptiveMovingAverage _frama;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _frama = FRAMA(_symbol, 20, 198);
    }

    public override void OnData(Slice data)
    {

        if (_frama.IsReady)
        {
            // The current value of _frama is represented by itself (_frama)
            // or _frama.Current.Value
            Plot("FractalAdaptiveMovingAverage", "frama", _frama);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `FractalAdaptiveMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

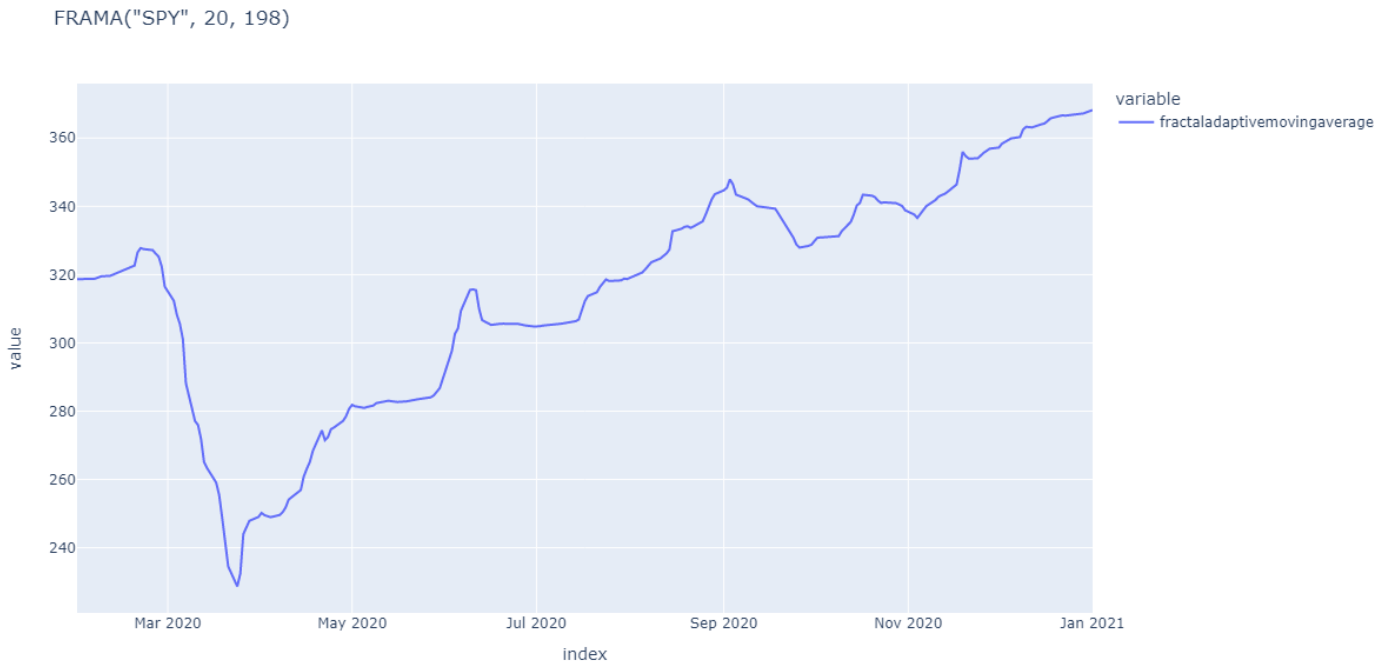
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class FractalAdaptiveMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private FractalAdaptiveMovingAverage _fractaladaptivemovingaverage;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _fractaladaptivemovingaverage = new FractalAdaptiveMovingAverage(20, 198);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _fractaladaptivemovingaverage.Update(bar);

        if (_fractaladaptivemovingaverage.IsReady)
        {
            // The current value of _fractaladaptivemovingaverage is represented by itself
(_fractaladaptivemovingaverage)
            // or _fractaladaptivemovingaverage.Current.Value
            Plot("FractalAdaptiveMovingAverage", "fractaladaptivemovingaverage", _fractaladaptivemovingaverage);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `FractalAdaptiveMovingAverage` indicator properties:



Indicator History

To get the historical data of the `FractalAdaptiveMovingAverage` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class FractalAdaptiveMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private FractalAdaptiveMovingAverage _frama;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _frama = FRAMA(_symbol, 20, 198);

        var indicatorHistory = IndicatorHistory(_frama, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_frama, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_frama, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Gamma

Introduction

Option Gamma indicator that calculate the gamma of an option

To view the implementation of this indicator, see the LEAN GitHub repository .

Using G Indicator

To create an automatic indicator for `Gamma` , call the `G` helper method from the `QCAlgorithm` class. The `G` method creates a `Gamma` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class GammaAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _option, _mirrorOption;
    private Gamma _g;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 20);

        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _option = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920C00564000");
        _mirrorOption = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920P00564000");
        AddOptionContract(_option, Resolution.Daily);
        AddOptionContract(_mirrorOption, Resolution.Daily);

        _g = G(_option, _mirrorOption);
    }

    public override void OnData(Slice data)
    {
        if (_g.IsReady)
        {
            // The current value of _g is represented by itself (_g)
            // or _g.Current.Value
            Plot("Gamma", "g", _g);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `Gamma` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

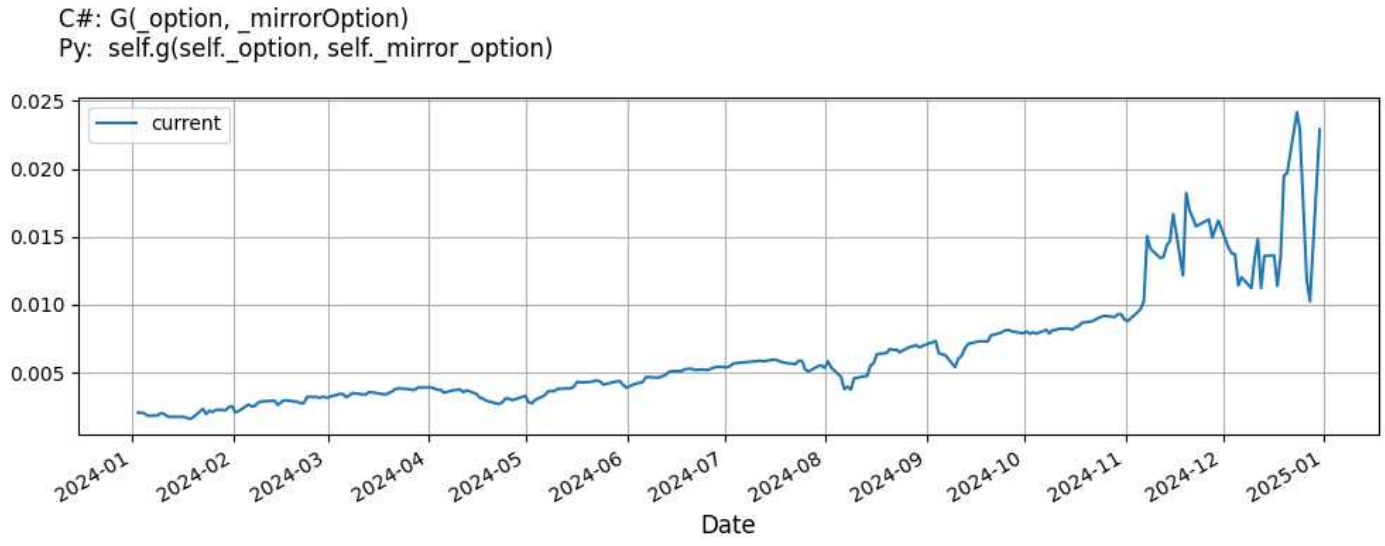
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class GammaAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _option, _mirrorOption;
    private Gamma _gamma;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 20);

        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        var interestRateModel = new InterestRateProvider();
        var dividendYieldModel = new DividendYieldProvider(_symbol);

        _option = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920C00564000");
        _mirrorOption = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920P00564000");
        AddOptionContract(_option, Resolution.Daily);
        AddOptionContract(_mirrorOption, Resolution.Daily);

        _gamma = new Gamma(_option, interestRateModel, dividendYieldModel, _mirrorOption);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _gamma.Update(new IndicatorDataPoint(_symbol, bar.EndTime, bar.Close));
        if (data.QuoteBars.TryGetValue(_option, out var quoteBar))
            _gamma.Update(new IndicatorDataPoint(_option, quoteBar.EndTime, quoteBar.Close));
        if (data.QuoteBars.TryGetValue(_mirrorOption, out quoteBar))
            _gamma.Update(new IndicatorDataPoint(_mirrorOption, quoteBar.EndTime, quoteBar.Close));

        if (_gamma.IsReady)
        {
            // The current value of _gamma is represented by itself (_gamma)
            // or _gamma.Current.Value
            Plot("Gamma", "gamma", _gamma);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `Gamma` indicator properties:



Indicator History

To get the historical data of the `Gamma` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory`

method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#

public class GammaAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _option, _mirrorOption;
    private Gamma _g;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 20);

        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _option = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920C00564000");
        _mirrorOption = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920P00564000");
        AddOptionContract(_option, Resolution.Daily);
        AddOptionContract(_mirrorOption, Resolution.Daily);

        _g = G(_option, _mirrorOption);

        var indicatorHistory = IndicatorHistory(_g, new[] { _symbol, _option, _mirrorOption }, 100,
Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_g, new[] { _symbol, _option, _mirrorOption },
TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_g, new[] { _symbol, _option, _mirrorOption }, new
DateTime(2024, 7, 1), new DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Heikin Ashi

Introduction

This indicator computes the Heikin-Ashi bar (HA) The Heikin-Ashi bar is calculated using the following formulas: HA_Close[0] = (Open[0] + High[0] + Low[0] + Close[0]) / 4 HA_Open[0] = (HA_Open[1] + HA_Close[1]) / 2 HA_High[0] = MAX(High[0], HA_Open[0], HA_Close[0]) HA_Low[0] = MIN(Low[0], HA_Open[0], HA_Close[0])

To view the implementation of this indicator, see the LEAN GitHub repository .

Using HeikinAshi Indicator

To create an automatic indicator for `HeikinAshi` , call the `HeikinAshi` helper method from the `QCAlgorithm` class. The `HeikinAshi` method creates a `HeikinAshi` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class HeikinAshiAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HeikinAshi _heikinashi;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _heikinashi = HeikinAshi(_symbol);
    }

    public override void OnData(Slice data)
    {

        if (_heikinashi.IsReady)
        {
            // The current value of _heikinashi is represented by itself (_heikinashi)
            // or _heikinashi.Current.Value
            Plot("HeikinAshi", "heikinashi", _heikinashi);
            // Plot all properties of abands
            Plot("HeikinAshi", "open", _heikinashi.Open);
            Plot("HeikinAshi", "high", _heikinashi.High);
            Plot("HeikinAshi", "low", _heikinashi.Low);
            Plot("HeikinAshi", "close", _heikinashi.Close);
            Plot("HeikinAshi", "volume", _heikinashi.Volume);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `HeikinAshi` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
```

```
public class HeikinAshiAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HeikinAshi _heikinashi;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _heikinashi = new HeikinAshi();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _heikinashi.Update(bar);

        if (_heikinashi.IsReady)
        {
            // The current value of _heikinashi is represented by itself (_heikinashi)
            // or _heikinashi.Current.Value
            Plot("HeikinAshi", "heikinashi", _heikinashi);
            // Plot all properties of abands
            Plot("HeikinAshi", "open", _heikinashi.Open);
            Plot("HeikinAshi", "high", _heikinashi.High);
            Plot("HeikinAshi", "low", _heikinashi.Low);
            Plot("HeikinAshi", "close", _heikinashi.Close);
            Plot("HeikinAshi", "volume", _heikinashi.Volume);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `HeikinAshi` indicator properties:

Indicator History

To get the historical data of the `HeikinAshi` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period

of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class HeikinAshiAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HeikinAshi _heikinashi;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _heikinashi = HeikinAshi(_symbol);

        var indicatorHistory = IndicatorHistory(_heikinashi, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_heikinashi, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_heikinashi, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var open = indicatorHistory.Select(x => ((dynamic)x).Open).ToList();
        var high = indicatorHistory.Select(x => ((dynamic)x).High).ToList();
        var low = indicatorHistory.Select(x => ((dynamic)x).Low).ToList();
        var close = indicatorHistory.Select(x => ((dynamic)x).Close).ToList();
        var volume = indicatorHistory.Select(x => ((dynamic)x).Volume).ToList();
    }
}
```

Supported Indicators

Hilbert Transform

Introduction

This indicator computes the Hilbert Transform Indicator by John Ehlers. By using present and prior price differences, and some feedback, price values are split into their complex number components of real (inPhase) and imaginary (quadrature) parts. source

To view the implementation of this indicator, see the LEAN GitHub repository .

Using HT Indicator

To create an automatic indicator for `HilbertTransform` , call the `HT` helper method from the `QCAlgorithm` class. The `HT` method creates a `HilbertTransform` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class HilbertTransformAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HilbertTransform _ht;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ht = HT(_symbol, 7, 0.635m, 0.338m);
    }

    public override void OnData(Slice data)
    {

        if (_ht.IsReady)
        {
            // The current value of _ht is represented by itself (_ht)
            // or _ht.Current.Value
            Plot("HilbertTransform", "ht", _ht);
            // Plot all properties of abands
            Plot("HilbertTransform", "inphase", _ht.InPhase);
            Plot("HilbertTransform", "quadrature", _ht.Quadrature);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `HilbertTransform` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class HilbertTransformAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HilbertTransform _hilberttransform;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _hilberttransform = new HilbertTransform(7, 0.635m, 0.338m);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _hilberttransform.Update(bar.EndTime, bar.Close);

        if (_hilberttransform.IsReady)
        {
            // The current value of _hilberttransform is represented by itself (_hilberttransform)
            // or _hilberttransform.Current.Value
            Plot("HilbertTransform", "hilberttransform", _hilberttransform);
            // Plot all properties of abands
            Plot("HilbertTransform", "inphase", _hilberttransform.InPhase);
            Plot("HilbertTransform", "quadrature", _hilberttransform.Quadrature);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `HilbertTransform` indicator properties:

Indicator History

To get the historical data of the `HilbertTransform` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class HilbertTransformAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HilbertTransform _ht;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ht = HT(_symbol, 7, 0.635m, 0.338m);

        var indicatorHistory = IndicatorHistory(_ht, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_ht, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_ht, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var inPhase = indicatorHistory.Select(x => ((dynamic)x).InPhase).ToList();
        var quadrature = indicatorHistory.Select(x => ((dynamic)x).Quadrature).ToList();
    }
}
```

Supported Indicators

Hull Moving Average

Introduction

Produces a Hull Moving Average as explained at http://www.alanhull.com/hull-moving-average/ and derived from the instructions for the Excel VBA code at http://finance4traders.blogspot.com/2009/06/how-to-calculate-hull-moving-average.html

To view the implementation of this indicator, see the LEAN GitHub repository .

Using HMA Indicator

To create an automatic indicator for `HullMovingAverage` , call the `HMA` helper method from the `QCAlgorithm` class. The `HMA` method creates a `HullMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class HullMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HullMovingAverage _hma;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _hma = HMA(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_hma.IsReady)
        {
            // The current value of _hma is represented by itself (_hma)
            // or _hma.Current.Value
            Plot("HullMovingAverage", "hma", _hma);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `HullMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

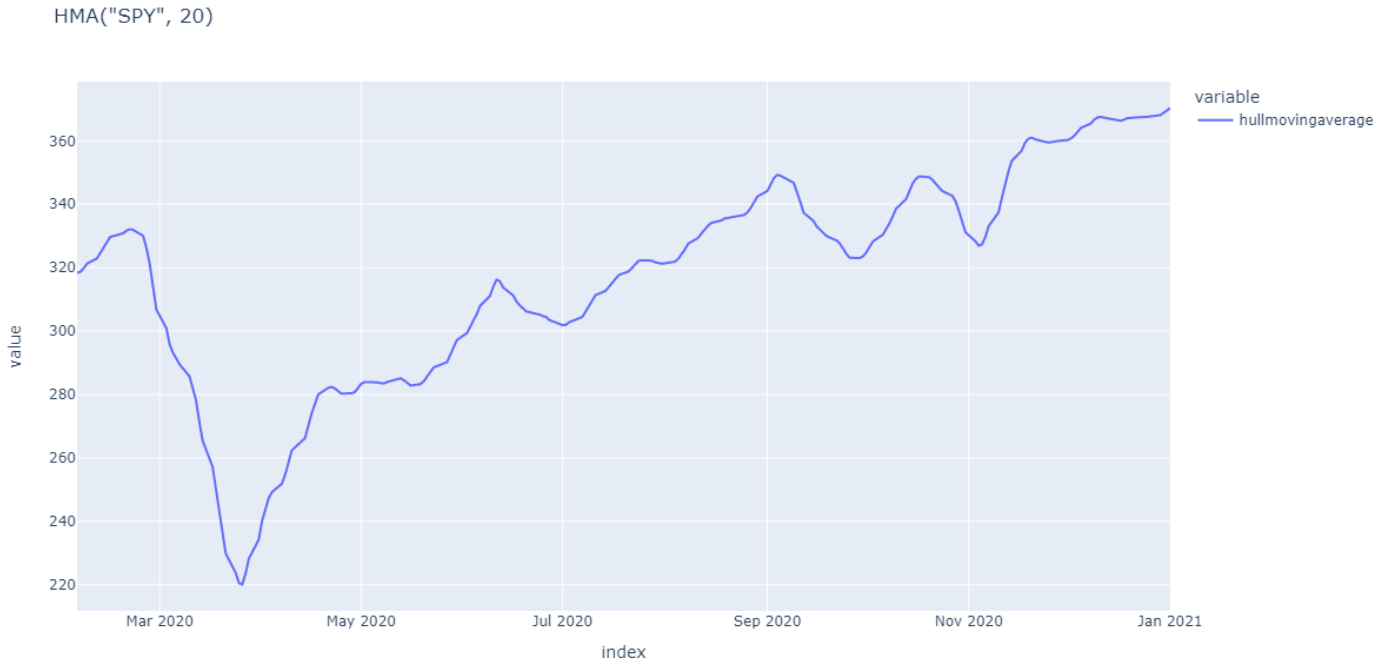
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class HullMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HullMovingAverage _hullmovingaverage;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _hullmovingaverage = new HullMovingAverage(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _hullmovingaverage.Update(bar.EndTime, bar.Close);

        if (_hullmovingaverage.IsReady)
        {
            // The current value of _hullmovingaverage is represented by itself (_hullmovingaverage)
            // or _hullmovingaverage.Current.Value
            Plot("HullMovingAverage", "hullmovingaverage", _hullmovingaverage);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `HullMovingAverage` indicator properties:



Indicator History

To get the historical data of the `HullMovingAverage` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class HullMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HullMovingAverage _hma;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _hma = HMA(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_hma, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_hma, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_hma, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Hurst Exponent

Introduction

This indicator represents the Hurst Exponent indicator, which is used to measure the long-term memory of a time series. - H less than 0.5: Mean-reverting; high values followed by low ones, stronger as H approaches 0. - H equal to 0.5: Random walk (geometric). - H greater than 0.5: Trending; high values followed by higher ones, stronger as H approaches 1.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using HE Indicator

To create an automatic indicator for `HurstExponent` , call the `HE` helper method from the `QCAlgorithm` class. The `HE` method creates a `HurstExponent` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class HurstExponentAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HurstExponent _he;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _he = HE(_symbol, 32);
    }

    public override void OnData(Slice data)
    {

        if (_he.IsReady)
        {
            // The current value of _he is represented by itself (_he)
            // or _he.Current.Value
            Plot("HurstExponent", "he", _he);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `HurstExponent` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

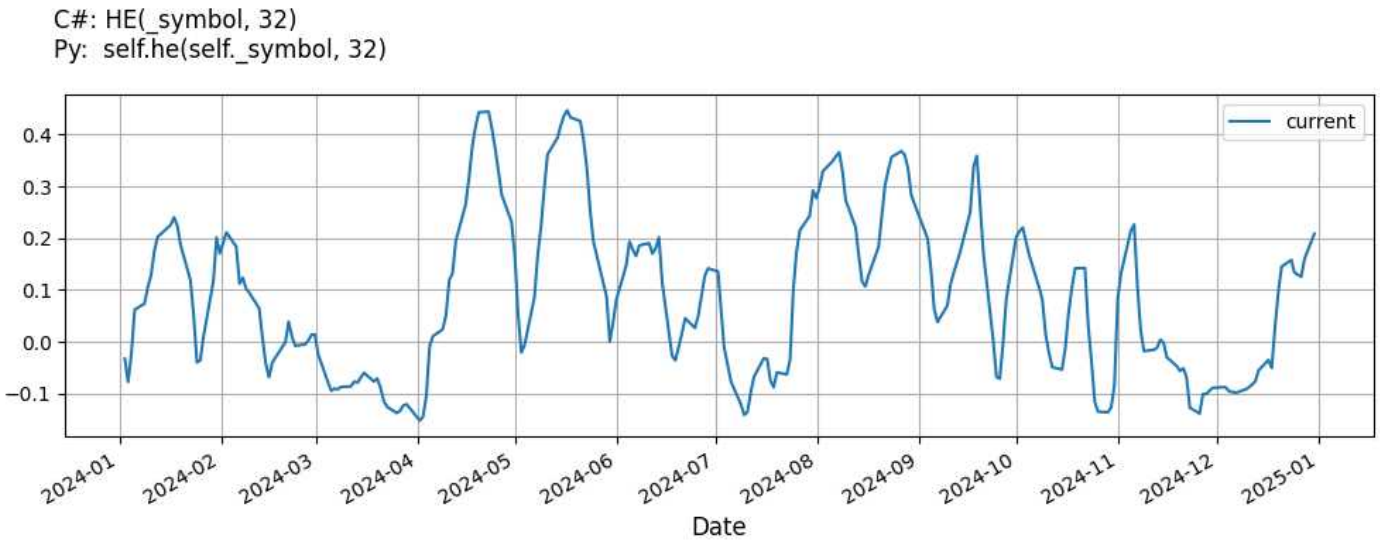
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class HurstExponentAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HurstExponent _hurstexponent;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _hurstexponent = new HurstExponent(32);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _hurstexponent.Update(bar.EndTime, bar.Close);

        if (_hurstexponent.IsReady)
        {
            // The current value of _hurstexponent is represented by itself (_hurstexponent)
            // or _hurstexponent.Current.Value
            Plot("HurstExponent", "hurstexponent", _hurstexponent);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `HurstExponent` indicator properties:



Indicator History

To get the historical data of the `HurstExponent` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class HurstExponentAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private HurstExponent _he;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _he = HE(_symbol, 32);

        var indicatorHistory = IndicatorHistory(_he, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_he, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_he, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Ichimoku Kinko Hyo

Introduction

This indicator computes the Ichimoku Kinko Hyo indicator. It consists of the following main indicators: Tenkan-sen: (Highest High + Lowest Low) / 2 for the specific period (normally 9) Kijun-sen: (Highest High + Lowest Low) / 2 for the specific period (normally 26) Senkou A Span: (Tenkan-sen + Kijun-sen)/ 2 from a specific number of periods ago (normally 26) Senkou B Span: (Highest High + Lowest Low) / 2 for the specific period (normally 52), from a specific number of periods ago (normally 26)

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ICHIMOKU Indicator

To create an automatic indicator for `IchimokuKinkoHyo` , call the `ICHIMOKU` helper method from the `QCAlgorithm` class. The `ICHIMOKU` method creates a `IchimokuKinkoHyo` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class IchimokuKinkoHyoAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private IchimokuKinkoHyo _ichimoku;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ichimoku = ICHIMOKU(_symbol, 9, 26, 17, 52, 26, 26);
    }

    public override void OnData(Slice data)
    {

        if (_ichimoku.IsReady)
        {
            // The current value of _ichimoku is represented by itself (_ichimoku)
            // or _ichimoku.Current.Value
            Plot("IchimokuKinkoHyo", "ichimoku", _ichimoku);
            // Plot all properties of abands
            Plot("IchimokuKinkoHyo", "tenkan", _ichimoku.Tenkan);
            Plot("IchimokuKinkoHyo", "kijun", _ichimoku.Kijun);
            Plot("IchimokuKinkoHyo", "senkoua", _ichimoku.SenkouA);
            Plot("IchimokuKinkoHyo", "senkoub", _ichimoku.SenkouB);
            Plot("IchimokuKinkoHyo", "chikou", _ichimoku.Chikou);
            Plot("IchimokuKinkoHyo", "tenkanmaximum", _ichimoku.TenkanMaximum);
            Plot("IchimokuKinkoHyo", "tenkanminimum", _ichimoku.TenkanMinimum);
            Plot("IchimokuKinkoHyo", "kijunmaximum", _ichimoku.KijunMaximum);
            Plot("IchimokuKinkoHyo", "kijunminimum", _ichimoku.KijunMinimum);
            Plot("IchimokuKinkoHyo", "senkoubmaximum", _ichimoku.SenkouBMaximum);
            Plot("IchimokuKinkoHyo", "senkoubminimum", _ichimoku.SenkouBMinimum);
            Plot("IchimokuKinkoHyo", "delayedtenkansenkoua", _ichimoku.DelayedTenkanSenkouA);
            Plot("IchimokuKinkoHyo", "delayedkijunsenkoua", _ichimoku.DelayedKijunSenkouA);
            Plot("IchimokuKinkoHyo", "delayedmaximumsenkoub", _ichimoku.DelayedMaximumSenkouB);
            Plot("IchimokuKinkoHyo", "delayedminimumsenkoub", _ichimoku.DelayedMinimumSenkouB);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `IchimokuKinkoHyo` indicator, so it doesn't automatically update. Manual indicators let you update

their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class IchimokuKinkoHyoAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private IchimokuKinkoHyo _ichimokukinkohyo;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ichimokukinkohyo = new IchimokuKinkoHyo(9, 26, 17, 52, 26, 26);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _ichimokukinkohyo.Update(bar);

        if (_ichimokukinkohyo.IsReady)
        {
            // The current value of _ichimokukinkohyo is represented by itself (_ichimokukinkohyo)
            // or _ichimokukinkohyo.Current.Value
            Plot("IchimokuKinkoHyo", "ichimokukinkohyo", _ichimokukinkohyo);
            // Plot all properties of abands
            Plot("IchimokuKinkoHyo", "tenkan", _ichimokukinkohyo.Tenkan);
            Plot("IchimokuKinkoHyo", "kijun", _ichimokukinkohyo.Kijun);
            Plot("IchimokuKinkoHyo", "senkoua", _ichimokukinkohyo.SenkouA);
            Plot("IchimokuKinkoHyo", "senkoub", _ichimokukinkohyo.SenkouB);
            Plot("IchimokuKinkoHyo", "chikou", _ichimokukinkohyo.Chikou);
            Plot("IchimokuKinkoHyo", "tenkanmaximum", _ichimokukinkohyo.TenkanMaximum);
            Plot("IchimokuKinkoHyo", "tenkanminimum", _ichimokukinkohyo.TenkanMinimum);
            Plot("IchimokuKinkoHyo", "kijunmaximum", _ichimokukinkohyo.KijunMaximum);
            Plot("IchimokuKinkoHyo", "kijunminimum", _ichimokukinkohyo.KijunMinimum);
            Plot("IchimokuKinkoHyo", "senkoubmaximum", _ichimokukinkohyo.SenkouBMaximum);
            Plot("IchimokuKinkoHyo", "senkoubminimum", _ichimokukinkohyo.SenkouBMinimum);
            Plot("IchimokuKinkoHyo", "delayedtenkansenkoua", _ichimokukinkohyo.DelayedTenkanSenkouA);
            Plot("IchimokuKinkoHyo", "delayedkijunsenkoua", _ichimokukinkohyo.DelayedKijunSenkouA);
            Plot("IchimokuKinkoHyo", "delayedmaximumsenkoub", _ichimokukinkohyo.DelayedMaximumSenkouB);
            Plot("IchimokuKinkoHyo", "delayedminimumsenkoub", _ichimokukinkohyo.DelayedMinimumSenkouB);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `IchimokuKinkoHyo` indicator properties:

Indicator History

To get the historical data of the `IchimokuKinkoHyo` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class IchimokuKinkoHyoAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private IchimokuKinkoHyo _ichimoku;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ichimoku = ICHIMOKU(_symbol, 9, 26, 17, 52, 26, 26);

        var indicatorHistory = IndicatorHistory(_ichimoku, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_ichimoku, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_ichimoku, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var tenkan = indicatorHistory.Select(x => ((dynamic)x).Tenkan).ToList();
        var kijun = indicatorHistory.Select(x => ((dynamic)x).Kijun).ToList();
        var senkouA = indicatorHistory.Select(x => ((dynamic)x).SenkouA).ToList();
        var senkouB = indicatorHistory.Select(x => ((dynamic)x).SenkouB).ToList();
        var chikou = indicatorHistory.Select(x => ((dynamic)x).Chikou).ToList();
        var tenkanMaximum = indicatorHistory.Select(x => ((dynamic)x).TenkanMaximum).ToList();
        var tenkanMinimum = indicatorHistory.Select(x => ((dynamic)x).TenkanMinimum).ToList();
        var kijunMaximum = indicatorHistory.Select(x => ((dynamic)x).KijunMaximum).ToList();
        var kijunMinimum = indicatorHistory.Select(x => ((dynamic)x).KijunMinimum).ToList();
        var senkouBMaximum = indicatorHistory.Select(x => ((dynamic)x).SenkouBMaximum).ToList();
        var senkouBMinimum = indicatorHistory.Select(x => ((dynamic)x).SenkouBMinimum).ToList();
        var delayedTenkanSenkouA = indicatorHistory.Select(x => ((dynamic)x).DelayedTenkanSenkouA).ToList();
        var delayedKijunSenkouA = indicatorHistory.Select(x => ((dynamic)x).DelayedKijunSenkouA).ToList();
        var delayedMaximumSenkouB = indicatorHistory.Select(x => ((dynamic)x).DelayedMaximumSenkouB).ToList();
        var delayedMinimumSenkouB = indicatorHistory.Select(x => ((dynamic)x).DelayedMinimumSenkouB).ToList();
    }
}
```

Supported Indicators

Identity

Introduction

This indicator represents an indicator that is a ready after ingesting a single sample and always returns the same value as it is given.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using Identity Indicator

To create an automatic indicator for `Identity` , call the `Identity` helper method from the `QCAlgorithm` class. The `Identity` method creates a `Identity` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class IdentityAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Identity _identity;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _identity = Identity(_symbol);
    }

    public override void OnData(Slice data)
    {

        if (_identity.IsReady)
        {
            // The current value of _identity is represented by itself (_identity)
            // or _identity.Current.Value
            Plot("Identity", "identity", _identity);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `Identity` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

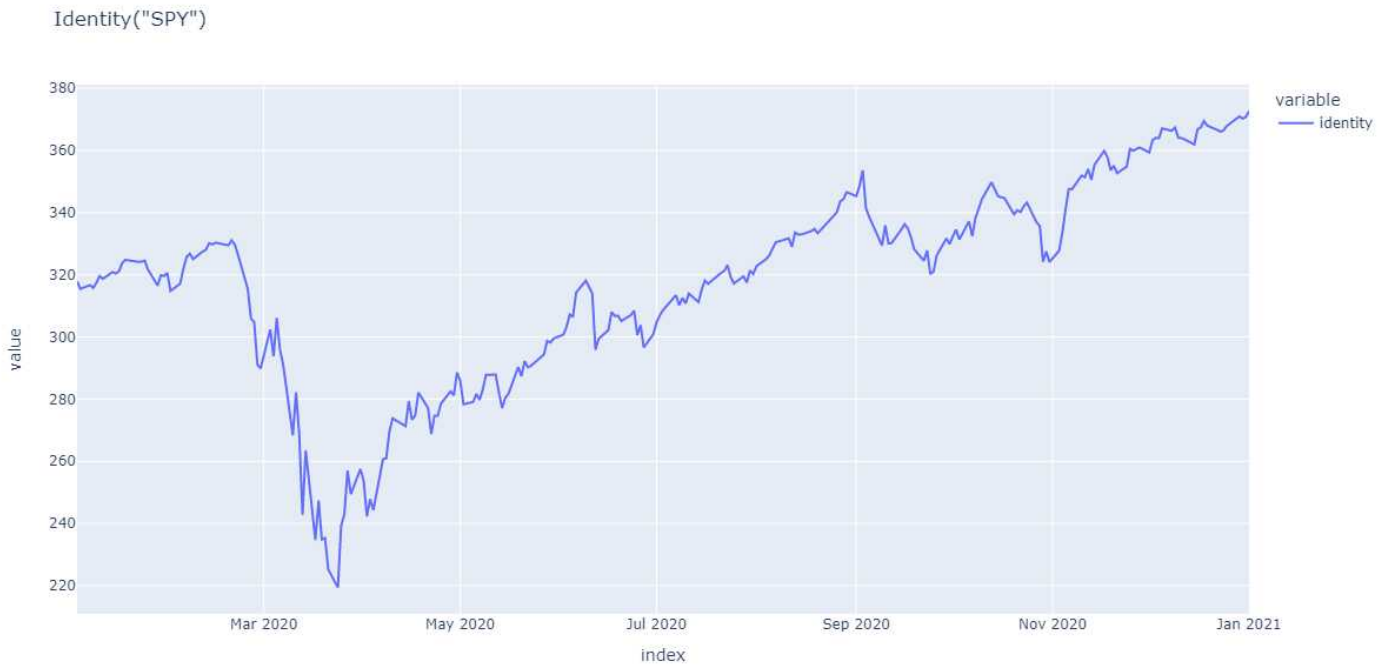
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class IdentityAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Identity _identity;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _identity = new Identity();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _identity.Update(bar.EndTime, bar.Close);

        if (_identity.IsReady)
        {
            // The current value of _identity is represented by itself (_identity)
            // or _identity.Current.Value
            Plot("Identity", "identity", _identity);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `Identity` indicator properties:



Indicator History

To get the historical data of the `Identity` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class IdentityAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Identity _identity;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _identity = Identity(_symbol);

        var indicatorHistory = IndicatorHistory(_identity, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_identity, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_identity, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Implied Volatility

Introduction

Implied Volatility indicator that calculate the IV of an option using Black-Scholes Model

To view the implementation of this indicator, see the LEAN GitHub repository .

Using IV Indicator

To create an automatic indicator for `ImpliedVolatility` , call the `IV` helper method from the `QCAlgorithm` class. The `IV` method creates a `ImpliedVolatility` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class ImpliedVolatilityAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _option, _mirrorOption;
    private ImpliedVolatility _iv;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 20);

        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _option = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920C00564000");
        _mirrorOption = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920P00564000");
        AddOptionContract(_option, Resolution.Daily);
        AddOptionContract(_mirrorOption, Resolution.Daily);

        _iv = IV(_option, _mirrorOption);
    }

    public override void OnData(Slice data)
    {
        if (_iv.IsReady)
        {
            // The current value of _iv is represented by itself (_iv)
            // or _iv.Current.Value
            Plot("ImpliedVolatility", "iv", _iv);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `ImpliedVolatility` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

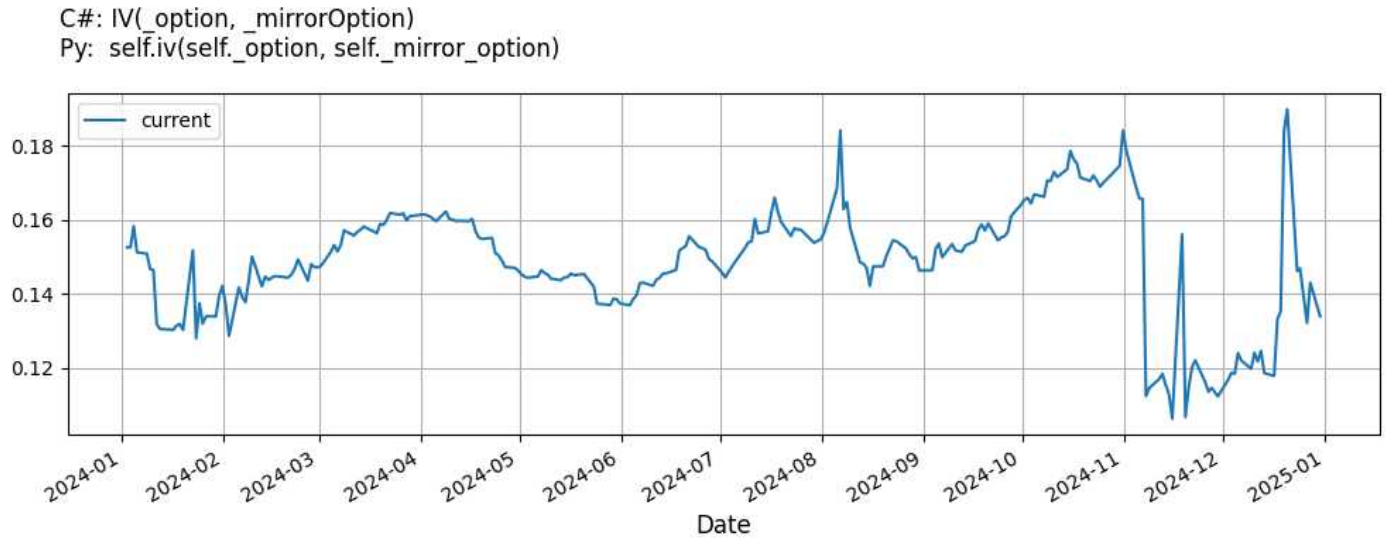
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ImpliedVolatilityAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _option, _mirrorOption;
    private ImpliedVolatility _impliedvolatility;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 20);

        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        var interestRateModel = new InterestRateProvider();
        var dividendYieldModel = new DividendYieldProvider(_symbol);

        _option = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920C00564000");
        _mirrorOption = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920P00564000");
        AddOptionContract(_option, Resolution.Daily);
        AddOptionContract(_mirrorOption, Resolution.Daily);

        _impliedvolatility = new ImpliedVolatility(_option, interestRateModel, dividendYieldModel, _mirrorOption);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _impliedvolatility.Update(new IndicatorDataPoint(_symbol, bar.EndTime, bar.Close));
        if (data.QuoteBars.TryGetValue(_option, out var quoteBar))
            _impliedvolatility.Update(new IndicatorDataPoint(_option, quoteBar.EndTime, quoteBar.Close));
        if (data.QuoteBars.TryGetValue(_mirrorOption, out quoteBar))
            _impliedvolatility.Update(new IndicatorDataPoint(_mirrorOption, quoteBar.EndTime, quoteBar.Close));

        if (_impliedvolatility.IsReady)
        {
            // The current value of _impliedvolatility is represented by itself (_impliedvolatility)
            // or _impliedvolatility.Current.Value
            Plot("ImpliedVolatility", "impliedvolatility", _impliedvolatility);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `ImpliedVolatility` indicator properties:



Indicator History

To get the historical data of the `ImpliedVolatility` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the

`IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class ImpliedVolatilityAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _option, _mirrorOption;
    private ImpliedVolatility _iv;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 20);

        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _option = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920C00564000");
        _mirrorOption = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920P00564000");
        AddOptionContract(_option, Resolution.Daily);
        AddOptionContract(_mirrorOption, Resolution.Daily);

        _iv = IV(_option, _mirrorOption);

        var indicatorHistory = IndicatorHistory(_iv, new[] { _symbol, _option, _mirrorOption }, 100,
Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_iv, new[] { _symbol, _option, _mirrorOption },
TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_iv, new[] { _symbol, _option, _mirrorOption }, new
DateTime(2024, 7, 1), new DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Internal Bar Strength

Introduction

The InternalBarStrenght indicator is a measure of the relative position of a period's closing price to the same period's high and low. The IBS can be interpreted to predict a bullish signal when displaying a low value and a bearish signal when presenting a high value.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using IBS Indicator

To create an automatic indicator for `InternalBarStrength` , call the `IBS` helper method from the `QCAlgorithm` class. The `IBS` method creates a `InternalBarStrength` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class InternalBarStrengthAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private InternalBarStrength _ibs;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ibs = IBS(_symbol);
    }

    public override void OnData(Slice data)
    {

        if (_ibs.IsReady)
        {
            // The current value of _ibs is represented by itself (_ibs)
            // or _ibs.Current.Value
            Plot("InternalBarStrength", "ibs", _ibs);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `InternalBarStrength` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

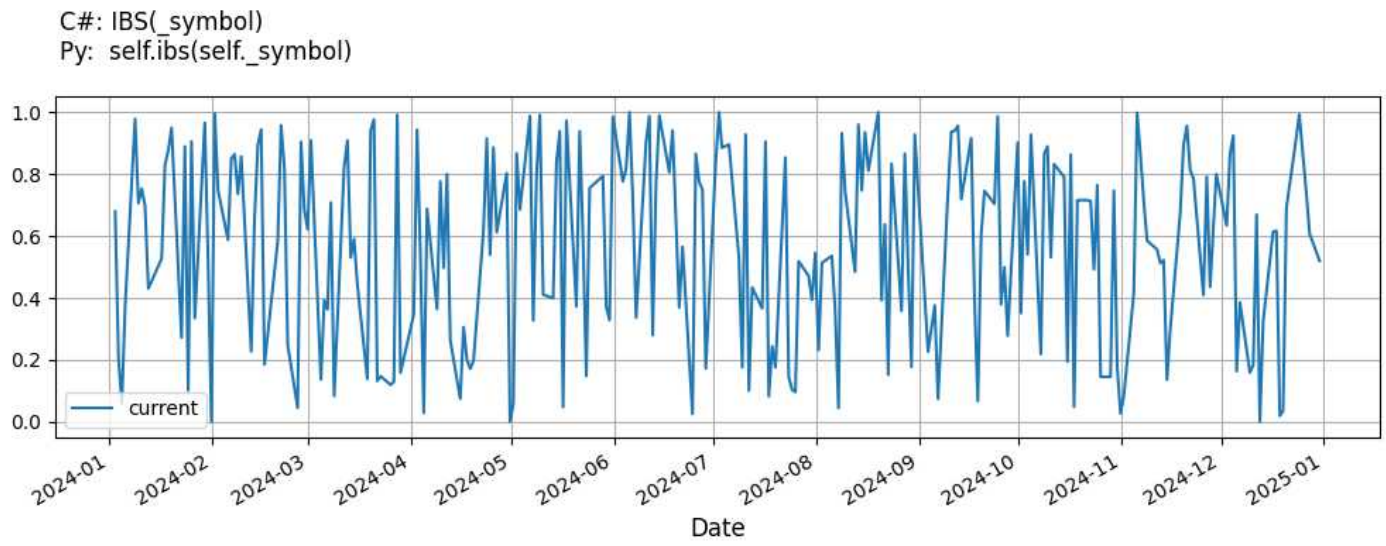
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class InternalBarStrengthAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private InternalBarStrength _internalbarstrength;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _internalbarstrength = new InternalBarStrength();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _internalbarstrength.Update(bar);

        if (_internalbarstrength.IsReady)
        {
            // The current value of _internalbarstrength is represented by itself (_internalbarstrength)
            // or _internalbarstrength.Current.Value
            Plot("InternalBarStrength", "internalbarstrength", _internalbarstrength);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `InternalBarStrength` indicator properties:



Indicator History

To get the historical data of the `InternalBarStrength` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class InternalBarStrengthAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private InternalBarStrength _ibs;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ibs = IBS(_symbol);

        var indicatorHistory = IndicatorHistory(_ibs, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_ibs, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_ibs, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Intraday Vwap

Introduction

Defines the canonical intraday VWAP indicator

To view the implementation of this indicator, see the LEAN GitHub repository .

Using VWAP Indicator

To create an automatic indicator for `IntradayVwap` , call the `VWAP` helper method from the `QCAlgorithm` class. The `VWAP` method creates a `IntradayVwap` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class IntradayVwapAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private IntradayVwap _vwap;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Minute).Symbol;
        _vwap = VWAP(_symbol);
    }

    public override void OnData(Slice data)
    {

        if (_vwap.IsReady)
        {
            // The current value of _vwap is represented by itself (_vwap)
            // or _vwap.Current.Value
            Plot("IntradayVwap", "vwap", _vwap);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `IntradayVwap` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class IntradayVwapAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private IntradayVwap _intradayvwap;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Minute).Symbol;
        _intradayvwap = new IntradayVwap();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _intradayvwap.Update(bar);

        if (_intradayvwap.IsReady)
        {
            // The current value of _intradayvwap is represented by itself (_intradayvwap)
            // or _intradayvwap.Current.Value
            Plot("IntradayVwap", "intradayvwap", _intradayvwap);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `IntradayVwap` indicator properties:

Indicator History

To get the historical data of the `IntradayVwap` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class IntradayVwapAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private IntradayVwap _vwap;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Minute).Symbol;
        _vwap = VWAP(_symbol);

        var indicatorHistory = IndicatorHistory(_vwap, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_vwap, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_vwap, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Kaufman Adaptive Moving Average

Introduction

This indicator computes the Kaufman Adaptive Moving Average (KAMA). The Kaufman Adaptive Moving Average is calculated as explained here: http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:kaufman_s_adaptive_moving_average

To view the implementation of this indicator, see the LEAN GitHub repository .

Using KAMA Indicator

To create an automatic indicator for `KaufmanAdaptiveMovingAverage` , call the `KAMA` helper method from the `QCAlgorithm` class. The `KAMA` method creates a `KaufmanAdaptiveMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

C#

```
public class KaufmanAdaptiveMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private KaufmanAdaptiveMovingAverage _kama;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _kama = KAMA(_symbol, 20, 10, 20);
    }

    public override void OnData(Slice data)
    {

        if (_kama.IsReady)
        {
            // The current value of _kama is represented by itself (_kama)
            // or _kama.Current.Value
            Plot("KaufmanAdaptiveMovingAverage", "kama", _kama);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `KaufmanAdaptiveMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

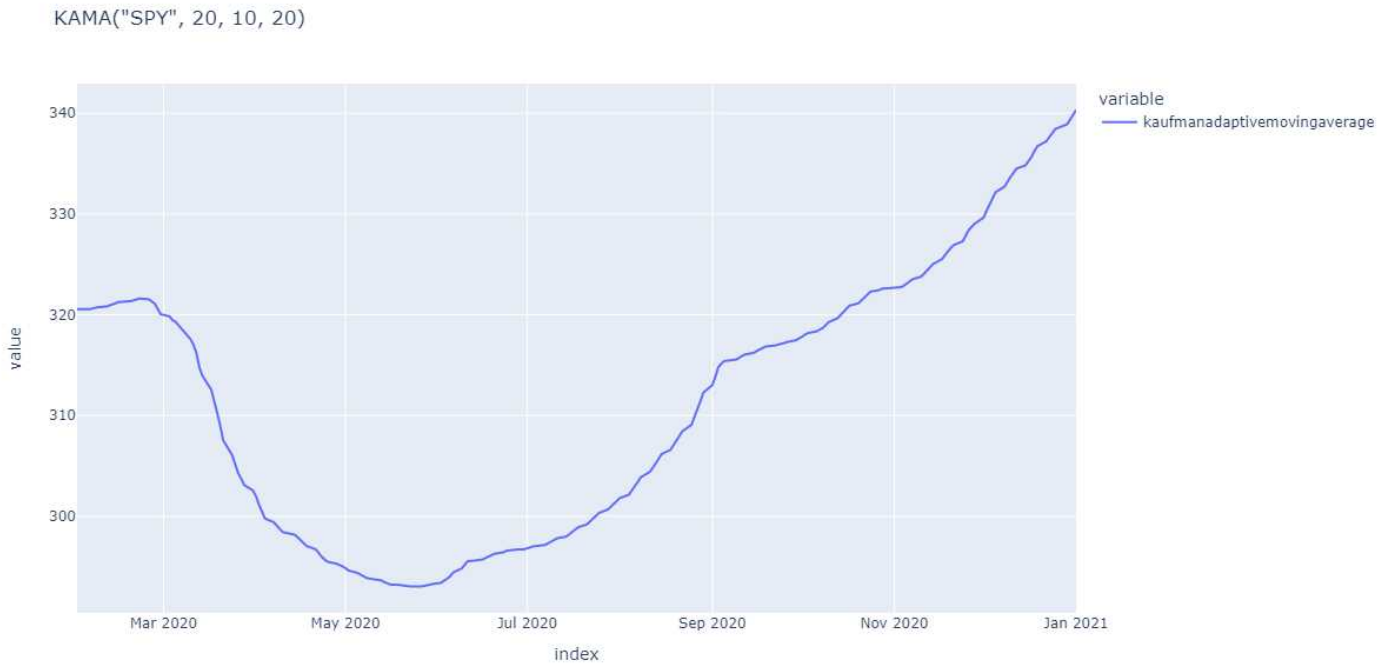
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class KaufmanAdaptiveMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private KaufmanAdaptiveMovingAverage _kaufmanadaptivemovingaverage;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _kaufmanadaptivemovingaverage = new KaufmanAdaptiveMovingAverage(20, 10, 20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _kaufmanadaptivemovingaverage.Update(bar.EndTime, bar.Close);

        if (_kaufmanadaptivemovingaverage.IsReady)
        {
            // The current value of _kaufmanadaptivemovingaverage is represented by itself
(_kaufmanadaptivemovingaverage)
            // or _kaufmanadaptivemovingaverage.Current.Value
            Plot("KaufmanAdaptiveMovingAverage", "kaufmanadaptivemovingaverage", _kaufmanadaptivemovingaverage);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `KaufmanAdaptiveMovingAverage` indicator properties:



Indicator History

To get the historical data of the `KaufmanAdaptiveMovingAverage` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class KaufmanAdaptiveMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private KaufmanAdaptiveMovingAverage _kama;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _kama = KAMA(_symbol, 20, 10, 20);

        var indicatorHistory = IndicatorHistory(_kama, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_kama, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_kama, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Kaufman Efficiency Ratio

Introduction

This indicator computes the Kaufman Efficiency Ratio (KER). The Kaufman Efficiency Ratio is calculated as explained here: https://www.marketvolume.com/technicalanalysis/efficiencyratio.asp

To view the implementation of this indicator, see the LEAN GitHub repository .

Using KER Indicator

To create an automatic indicator for `KaufmanEfficiencyRatio` , call the `KER` helper method from the `QCAlgorithm` class. The `KER` method creates a `KaufmanEfficiencyRatio` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class KaufmanEfficiencyRatioAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private KaufmanEfficiencyRatio _ker;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ker = KER(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_ker.IsReady)
        {
            // The current value of _ker is represented by itself (_ker)
            // or _ker.Current.Value
            Plot("KaufmanEfficiencyRatio", "ker", _ker);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `KaufmanEfficiencyRatio` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

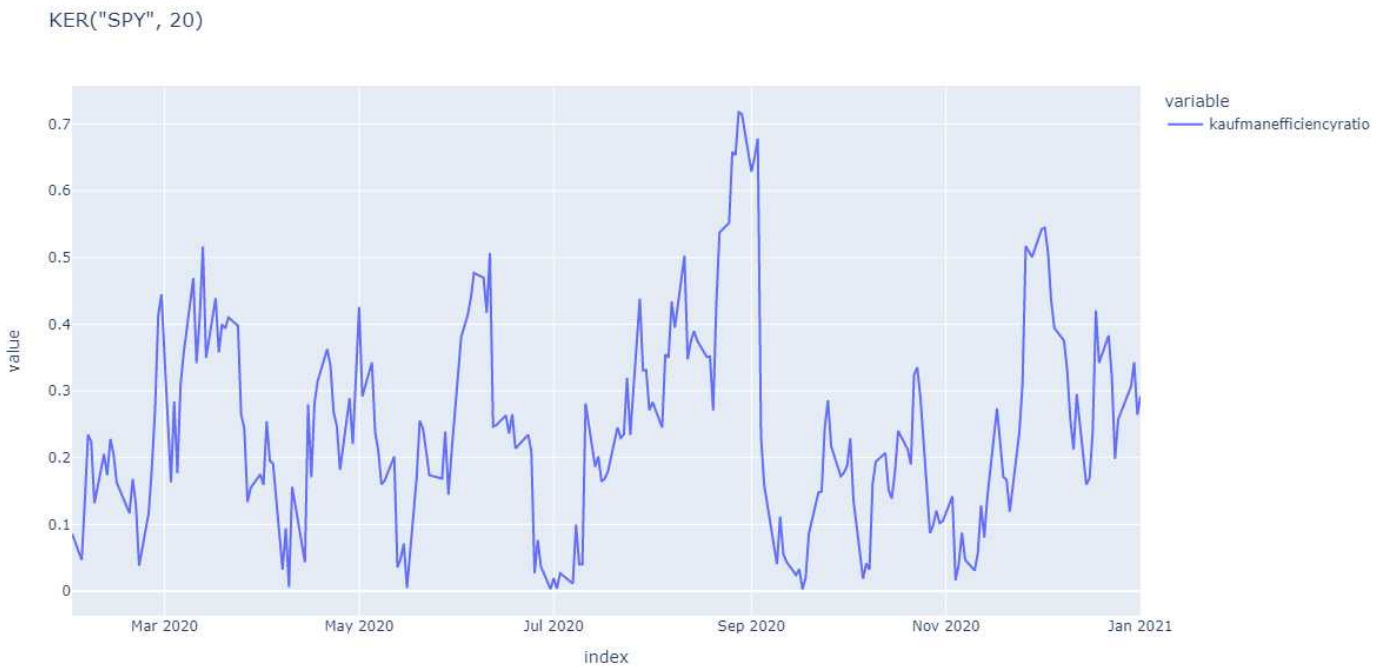
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class KaufmanEfficiencyRatioAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private KaufmanEfficiencyRatio _kaufmanefficiencyratio;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _kaufmanefficiencyratio = new KaufmanEfficiencyRatio(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _kaufmanefficiencyratio.Update(bar.EndTime, bar.Close);

        if (_kaufmanefficiencyratio.IsReady)
        {
            // The current value of _kaufmanefficiencyratio is represented by itself (_kaufmanefficiencyratio)
            // or _kaufmanefficiencyratio.Current.Value
            Plot("KaufmanEfficiencyRatio", "kaufmanefficiencyratio", _kaufmanefficiencyratio);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `KaufmanEfficiencyRatio` indicator properties:



Indicator History

To get the historical data of the `KaufmanEfficiencyRatio` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class KaufmanEfficiencyRatioAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private KaufmanEfficiencyRatio _ker;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ker = KER(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_ker, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_ker, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_ker, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Keltner Channels

Introduction

This indicator creates a moving average (middle band) with an upper band and lower band fixed at k average True range multiples away from the middle band.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using KCH Indicator

To create an automatic indicator for `KeltnerChannels` , call the `KCH` helper method from the `QCAlgorithm` class. The `KCH` method creates a `KeltnerChannels` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class KeltnerChannelsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private KeltnerChannels _kch;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _kch = KCH(_symbol, 20, 2, MovingAverageType.Simple);
    }

    public override void OnData(Slice data)
    {

        if (_kch.IsReady)
        {
            // The current value of _kch is represented by itself (_kch)
            // or _kch.Current.Value
            Plot("KeltnerChannels", "kch", _kch);
            // Plot all properties of abands
            Plot("KeltnerChannels", "middleband", _kch.MiddleBand);
            Plot("KeltnerChannels", "upperband", _kch.UpperBand);
            Plot("KeltnerChannels", "lowerband", _kch.LowerBand);
            Plot("KeltnerChannels", "averagetruerange", _kch.AverageTrueRange);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `KeltnerChannels` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

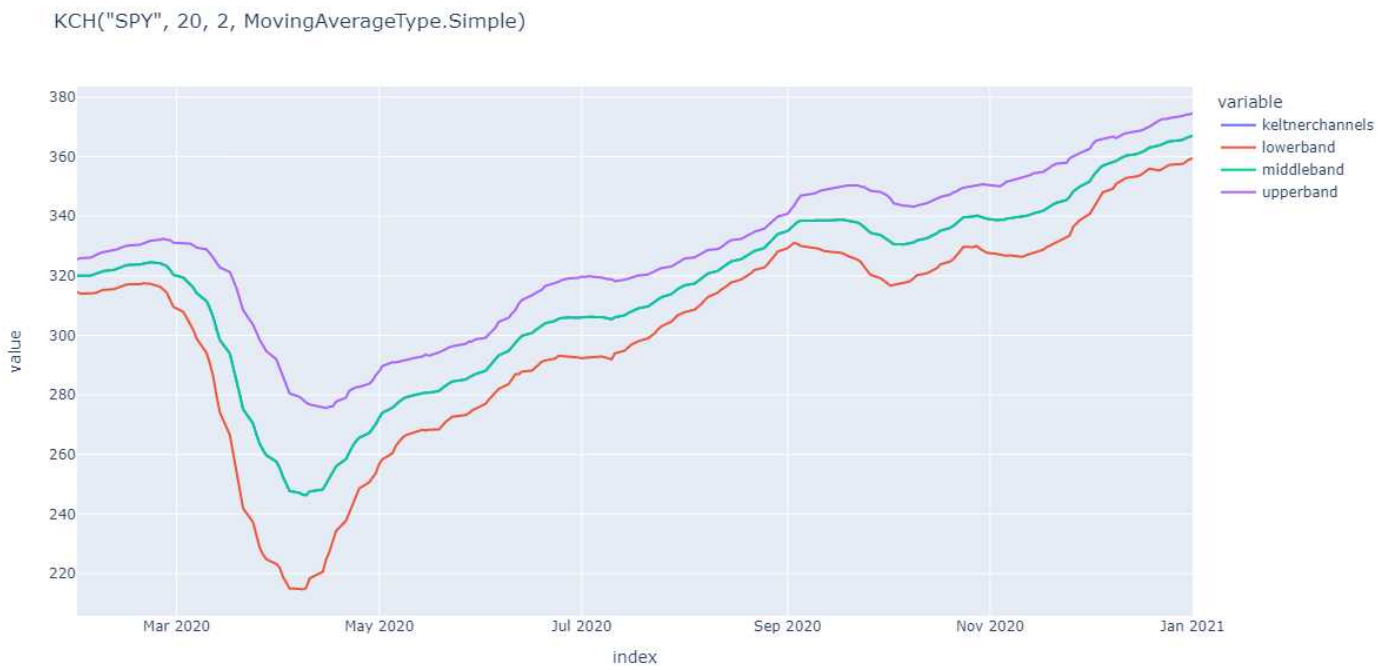
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class KeltnerChannelsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private KeltnerChannels _keltnerchannels;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _keltnerchannels = new KeltnerChannels(20, 2, MovingAverageType.Simple);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _keltnerchannels.Update(bar);

        if (_keltnerchannels.IsReady)
        {
            // The current value of _keltnerchannels is represented by itself (_keltnerchannels)
            // or _keltnerchannels.Current.Value
            Plot("KeltnerChannels", "keltnerchannels", _keltnerchannels);
            // Plot all properties of abands
            Plot("KeltnerChannels", "middleband", _keltnerchannels.MiddleBand);
            Plot("KeltnerChannels", "upperband", _keltnerchannels.UpperBand);
            Plot("KeltnerChannels", "lowerband", _keltnerchannels.LowerBand);
            Plot("KeltnerChannels", "averagetruerange", _keltnerchannels.AverageTrueRange);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `KeltnerChannels` indicator properties:



Indicator History

To get the historical data of the `KeltnerChannels` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class KeltnerChannelsAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private KeltnerChannels _kch;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _kch = KCH(_symbol, 20, 2, MovingAverageType.Simple);

        var indicatorHistory = IndicatorHistory(_kch, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_kch, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_kch, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var middleBand = indicatorHistory.Select(x => ((dynamic)x).MiddleBand).ToList();
        var upperBand = indicatorHistory.Select(x => ((dynamic)x).UpperBand).ToList();
        var lowerBand = indicatorHistory.Select(x => ((dynamic)x).LowerBand).ToList();
        var averageTrueRange = indicatorHistory.Select(x => ((dynamic)x).AverageTrueRange).ToList();
    }
}
```

Supported Indicators

Klinger Volume Oscillator

Introduction

The Klinger Volume Oscillator (KVO) is a technical indicator that analyzes the relationship between price movement and trading volume to assess the strength of market trends and identify potential trend reversals. As a volume-based oscillator, it measures the force behind price movements by incorporating volume data adjusted for price trends and specific conditions. Traders use the KVO to analyze its behavior relative to price action, looking for patterns such as divergences or crossovers that can provide insights into market trends and potential turning points.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using KVO Indicator

To create an automatic indicator for `KlingerVolumeOscillator` , call the `KVO` helper method from the `QCAlgorithm` class. The `KVO` method creates a `KlingerVolumeOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class KlingerVolumeOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private KlingerVolumeOscillator _kvo;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _kvo = KVO(_symbol, 5, 10, 13);
    }

    public override void OnData(Slice data)
    {

        if (_kvo.IsReady)
        {
            // The current value of _kvo is represented by itself (_kvo)
            // or _kvo.Current.Value
            Plot("KlingerVolumeOscillator", "kvo", _kvo);
            // Plot all properties of abands
            Plot("KlingerVolumeOscillator", "signal", _kvo.Signal);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `KlingerVolumeOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

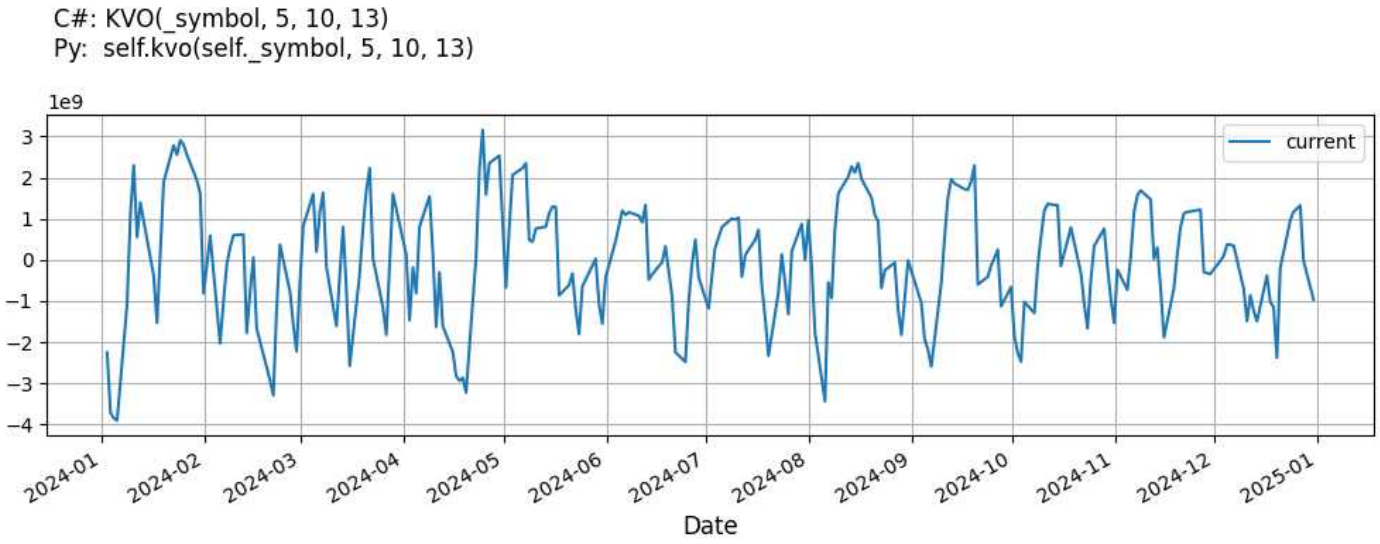
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class KlingerVolumeOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private KlingerVolumeOscillator _klingervolumeoscillator;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _klingervolumeoscillator = new KlingerVolumeOscillator(5, 10, 13);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _klingervolumeoscillator.Update(bar);

        if (_klingervolumeoscillator.IsReady)
        {
            // The current value of _klingervolumeoscillator is represented by itself (_klingervolumeoscillator)
            // or _klingervolumeoscillator.Current.Value
            Plot("KlingerVolumeOscillator", "klingervolumeoscillator", _klingervolumeoscillator);
            // Plot all properties of abands
            Plot("KlingerVolumeOscillator", "signal", _klingervolumeoscillator.Signal);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `KlingerVolumeOscillator` indicator properties:



Indicator History

To get the historical data of the `KlingerVolumeOscillator` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class KlingerVolumeOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private KlingerVolumeOscillator _kvo;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _kvo = KVO(_symbol, 5, 10, 13);

        var indicatorHistory = IndicatorHistory(_kvo, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_kvo, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_kvo, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var signal = indicatorHistory.Select(x => ((dynamic)x).Signal).ToList();
    }
}
```

Supported Indicators

Know Sure Thing

Introduction

This indicator creates a moving average (middle band) with an upper band and lower band fixed at k standard deviations above and below the moving average.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using KST Indicator

To create an automatic indicator for `KnowSureThing` , call the `KST` helper method from the `QCAlgorithm` class. The `KST` method creates a `KnowSureThing` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class KnowSureThingAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private KnowSureThing _kst;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _kst = KST(_symbol, 5, 5, 10, 5, 15, 5, 25, 10, 9, MovingAverageType.Simple);
    }

    public override void OnData(Slice data)
    {

        if (_kst.IsReady)
        {
            // The current value of _kst is represented by itself (_kst)
            // or _kst.Current.Value
            Plot("KnowSureThing", "kst", _kst);
            // Plot all properties of abands
            Plot("KnowSureThing", "roc1", _kst.ROC1);
            Plot("KnowSureThing", "roc2", _kst.ROC2);
            Plot("KnowSureThing", "roc3", _kst.ROC3);
            Plot("KnowSureThing", "roc4", _kst.ROC4);
            Plot("KnowSureThing", "roc1ma", _kst.ROC1MA);
            Plot("KnowSureThing", "roc2ma", _kst.ROC2MA);
            Plot("KnowSureThing", "roc3ma", _kst.ROC3MA);
            Plot("KnowSureThing", "roc4ma", _kst.ROC4MA);
            Plot("KnowSureThing", "signalline", _kst.SignalLine);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `KnowSureThing` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

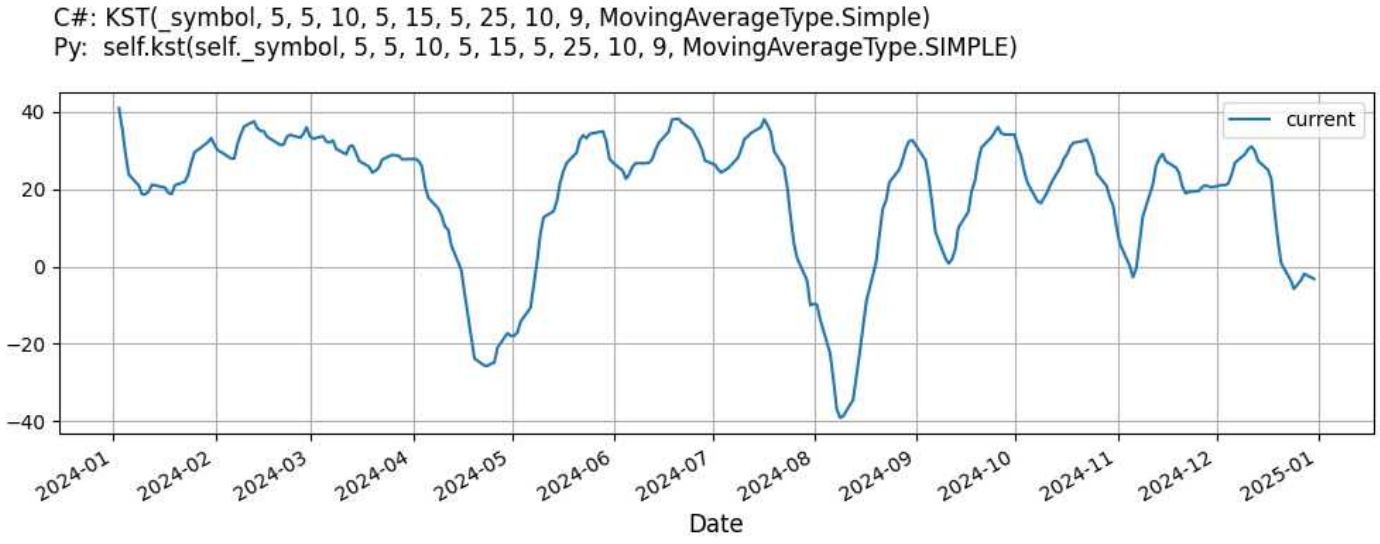
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class KnowSureThingAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private KnowSureThing _knowsurething;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _knowsurething = new KnowSureThing(5, 5, 10, 5, 15, 5, 25, 10, 9, MovingAverageType.Simple);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _knowsurething.Update(bar.EndTime, bar.Close);

        if (_knowsurething.IsReady)
        {
            // The current value of _knowsurething is represented by itself (_knowsurething)
            // or _knowsurething.Current.Value
            Plot("KnowSureThing", "knowsurething", _knowsurething);
            // Plot all properties of abands
            Plot("KnowSureThing", "roc1", _knowsurething.ROC1);
            Plot("KnowSureThing", "roc2", _knowsurething.ROC2);
            Plot("KnowSureThing", "roc3", _knowsurething.ROC3);
            Plot("KnowSureThing", "roc4", _knowsurething.ROC4);
            Plot("KnowSureThing", "roc1ma", _knowsurething.ROC1MA);
            Plot("KnowSureThing", "roc2ma", _knowsurething.ROC2MA);
            Plot("KnowSureThing", "roc3ma", _knowsurething.ROC3MA);
            Plot("KnowSureThing", "roc4ma", _knowsurething.ROC4MA);
            Plot("KnowSureThing", "signalline", _knowsurething.SignalLine);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `KnowSureThing` indicator properties:



Indicator History

To get the historical data of the `KnowSureThing` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class KnowSureThingAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private KnowSureThing _kst;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _kst = KST(_symbol, 5, 5, 10, 5, 15, 5, 25, 10, 9, MovingAverageType.Simple);

        var indicatorHistory = IndicatorHistory(_kst, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_kst, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_kst, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var roc1 = indicatorHistory.Select(x => ((dynamic)x).ROC1).ToList();
        var roc2 = indicatorHistory.Select(x => ((dynamic)x).ROC2).ToList();
        var roc3 = indicatorHistory.Select(x => ((dynamic)x).ROC3).ToList();
        var roc4 = indicatorHistory.Select(x => ((dynamic)x).ROC4).ToList();
        var roc1ma = indicatorHistory.Select(x => ((dynamic)x).ROC1MA).ToList();
        var roc2ma = indicatorHistory.Select(x => ((dynamic)x).ROC2MA).ToList();
        var roc3ma = indicatorHistory.Select(x => ((dynamic)x).ROC3MA).ToList();
        var roc4ma = indicatorHistory.Select(x => ((dynamic)x).ROC4MA).ToList();
        var signalLine = indicatorHistory.Select(x => ((dynamic)x).SignalLine).ToList();
    }
}
```

Supported Indicators

Least Squares Moving Average

Introduction

The Least Squares Moving Average (LSMA) first calculates a least squares regression line over the preceding time periods, and then projects it forward to the current period. In essence, it calculates what the value would be if the regression line continued. source

To view the implementation of this indicator, see the LEAN GitHub repository .

Using LSMA Indicator

To create an automatic indicator for `LeastSquaresMovingAverage` , call the `LSMA` helper method from the `QCAlgorithm` class. The `LSMA` method creates a `LeastSquaresMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class LeastSquaresMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private LeastSquaresMovingAverage _lsma;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _lsma = LSMA(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_lsma.IsReady)
        {
            // The current value of _lsma is represented by itself (_lsma)
            // or _lsma.Current.Value
            Plot("LeastSquaresMovingAverage", "lsma", _lsma);
            // Plot all properties of abands
            Plot("LeastSquaresMovingAverage", "intercept", _lsma.Intercept);
            Plot("LeastSquaresMovingAverage", "slope", _lsma.Slope);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `LeastSquaresMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class LeastSquaresMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private LeastSquaresMovingAverage _leastsquaresmovingaverage;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _leastsquaresmovingaverage = new LeastSquaresMovingAverage(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _leastsquaresmovingaverage.Update(bar.EndTime, bar.Close);

        if (_leastsquaresmovingaverage.IsReady)
        {
            // The current value of _leastsquaresmovingaverage is represented by itself (_leastsquaresmovingaverage)
            // or _leastsquaresmovingaverage.Current.Value
            Plot("LeastSquaresMovingAverage", "leastsquaresmovingaverage", _leastsquaresmovingaverage);
            // Plot all properties of abands
            Plot("LeastSquaresMovingAverage", "intercept", _leastsquaresmovingaverage.Intercept);
            Plot("LeastSquaresMovingAverage", "slope", _leastsquaresmovingaverage.Slope);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `LeastSquaresMovingAverage` indicator properties:

Indicator History

To get the historical data of the `LeastSquaresMovingAverage` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class LeastSquaresMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private LeastSquaresMovingAverage _lsma;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _lsma = LSMA(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_lsma, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_lsma, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_lsma, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var intercept = indicatorHistory.Select(x => ((dynamic)x).Intercept).ToList();
        var slope = indicatorHistory.Select(x => ((dynamic)x).Slope).ToList();
    }
}
```

Supported Indicators

Linear Weighted Moving Average

Introduction

This indicator represents the traditional Weighted Moving Average indicator. The weight are linearly distributed according to the number of periods in the indicator. For example, a 4 period indicator will have a numerator of (4 * window[0]) + (3 * window[1]) + (2 * window[2]) + window[3] and a denominator of 4 + 3 + 2 + 1 = 10 During the warm up period, IsReady will return False, but the LWMA will still be computed correctly because the denominator will be the minimum of Samples factorial or Size factorial and the computation iterates over that minimum value. The RollingWindow of inputs is created when the indicator is created. A RollingWindow of LWMAs is not saved. That is up to the caller.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using LWMA Indicator

To create an automatic indicator for `LinearWeightedMovingAverage` , call the `LWMA` helper method from the `QCAlgorithm` class. The `LWMA` method creates a `LinearWeightedMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class LinearWeightedMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private LinearWeightedMovingAverage _lwma;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _lwma = LWMA(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_lwma.IsReady)
        {
            // The current value of _lwma is represented by itself (_lwma)
            // or _lwma.Current.Value
            Plot("LinearWeightedMovingAverage", "lwma", _lwma);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `LinearWeightedMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

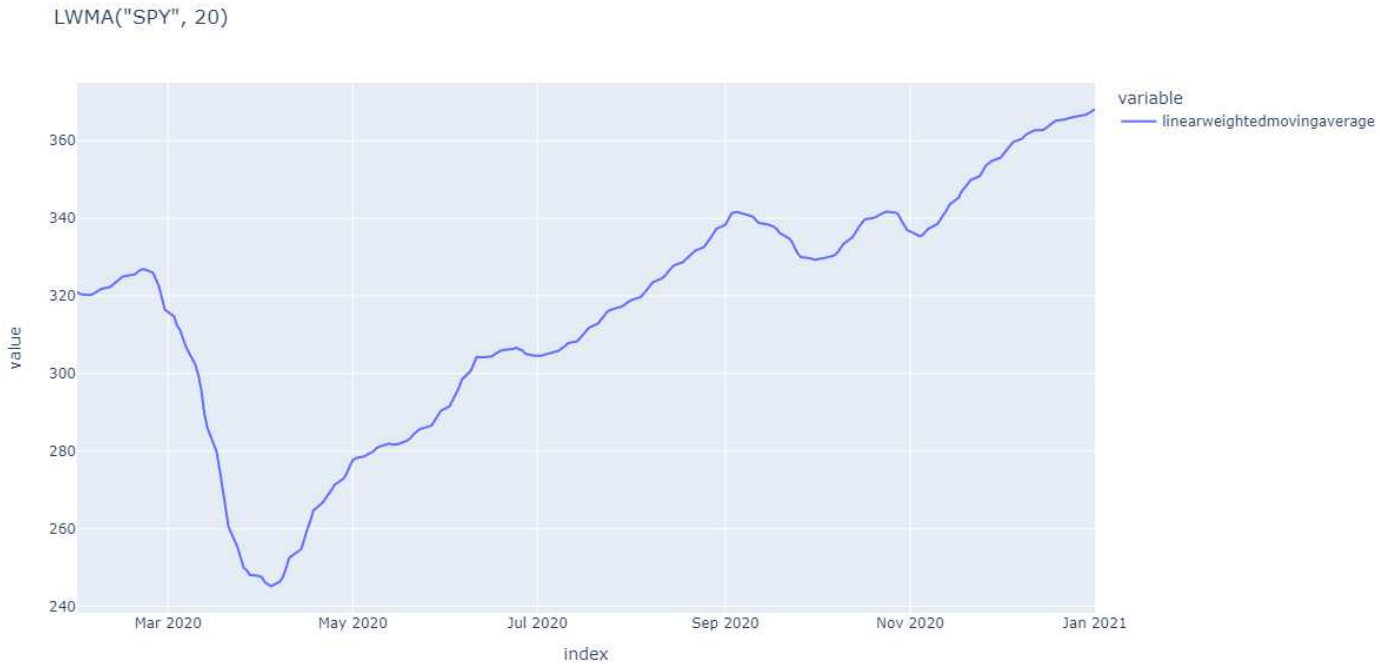
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class LinearWeightedMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private LinearWeightedMovingAverage _linearweightedmovingaverage;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _linearweightedmovingaverage = new LinearWeightedMovingAverage(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _linearweightedmovingaverage.Update(bar.EndTime, bar.Close);

        if (_linearweightedmovingaverage.IsReady)
        {
            // The current value of _linearweightedmovingaverage is represented by itself
(_linearweightedmovingaverage)
            // or _linearweightedmovingaverage.Current.Value
            Plot("LinearWeightedMovingAverage", "linearweightedmovingaverage", _linearweightedmovingaverage);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `LinearWeightedMovingAverage` indicator properties:



Indicator History

To get the historical data of the `LinearWeightedMovingAverage` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class LinearWeightedMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private LinearWeightedMovingAverage _lwma;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _lwma = LWMA(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_lwma, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_lwma, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_lwma, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Log Return

Introduction

This indicator represents the LogReturn indicator (LOGR) - log returns are useful for identifying price convergence/divergence in a given period - logr = log (current price / last price in period)

To view the implementation of this indicator, see the LEAN GitHub repository .

Using LOGR Indicator

To create an automatic indicator for `LogReturn` , call the `LOGR` helper method from the `QCAlgorithm` class. The `LOGR` method creates a `LogReturn` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class LogReturnAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private LogReturn _logr;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _logr = LOGR(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_logr.IsReady)
        {
            // The current value of _logr is represented by itself (_logr)
            // or _logr.Current.Value
            Plot("LogReturn", "logr", _logr);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `LogReturn` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class LogReturnAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private LogReturn _logreturn;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _logreturn = new LogReturn(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _logreturn.Update(bar.EndTime, bar.Close);

        if (_logreturn.IsReady)
        {
            // The current value of _logreturn is represented by itself (_logreturn)
            // or _logreturn.Current.Value
            Plot("LogReturn", "logreturn", _logreturn);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `LogReturn` indicator properties:

Indicator History

To get the historical data of the `LogReturn` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class LogReturnAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private LogReturn _logr;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _logr = LOGR(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_logr, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_logr, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_logr, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Mass Index

Introduction

The Mass Index uses the high-low range to identify trend reversals based on range expansions. In this sense, the Mass Index is a volatility indicator that does not have a directional bias. Instead, the Mass Index identifies range bulges that can foreshadow a reversal of the current trend. Developed by Donald Dorsey.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MASS Indicator

To create an automatic indicator for `MassIndex` , call the `MASS` helper method from the `QCAlgorithm` class. The `MASS` method creates a `MassIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class MassIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MassIndex _mass;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _mass = MASS(_symbol, 9, 25);
    }

    public override void OnData(Slice data)
    {

        if (_mass.IsReady)
        {
            // The current value of _mass is represented by itself (_mass)
            // or _mass.Current.Value
            Plot("MassIndex", "mass", _mass);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `MassIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

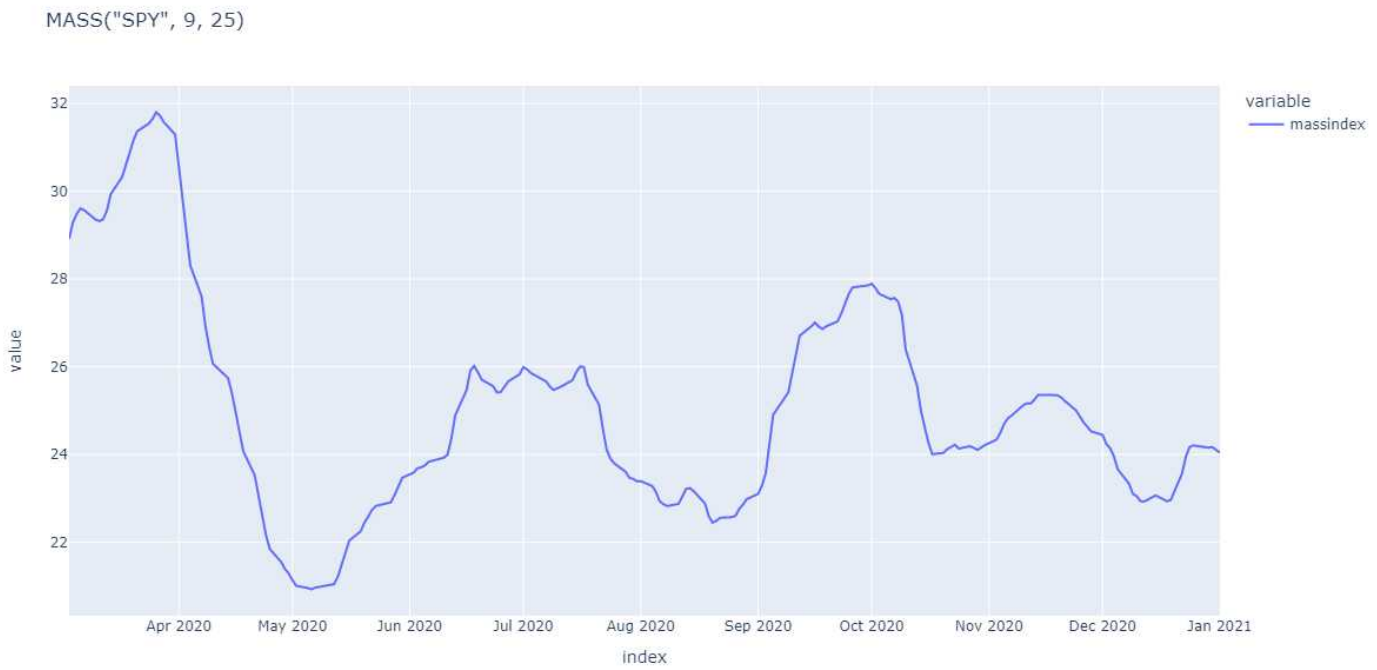
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class MassIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MassIndex _massindex;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _massindex = new MassIndex(9, 25);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _massindex.Update(bar);

        if (_massindex.IsReady)
        {
            // The current value of _massindex is represented by itself (_massindex)
            // or _massindex.Current.Value
            Plot("MassIndex", "massindex", _massindex);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `MassIndex` indicator properties:



Indicator History

To get the historical data of the `MassIndex` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class MassIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MassIndex _mass;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _mass = MASS(_symbol, 9, 25);

        var indicatorHistory = IndicatorHistory(_mass, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_mass, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_mass, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Maximum

Introduction

This indicator represents an indicator capable of tracking the maximum value and how many periods ago it occurred

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MAX Indicator

To create an automatic indicator for `Maximum` , call the `MAX` helper method from the `QCAlgorithm` class. The `MAX` method creates a `Maximum` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class MaximumAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Maximum _max;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _max = MAX(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_max.IsReady)
        {
            // The current value of _max is represented by itself (_max)
            // or _max.Current.Value
            Plot("Maximum", "max", _max);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `Maximum` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

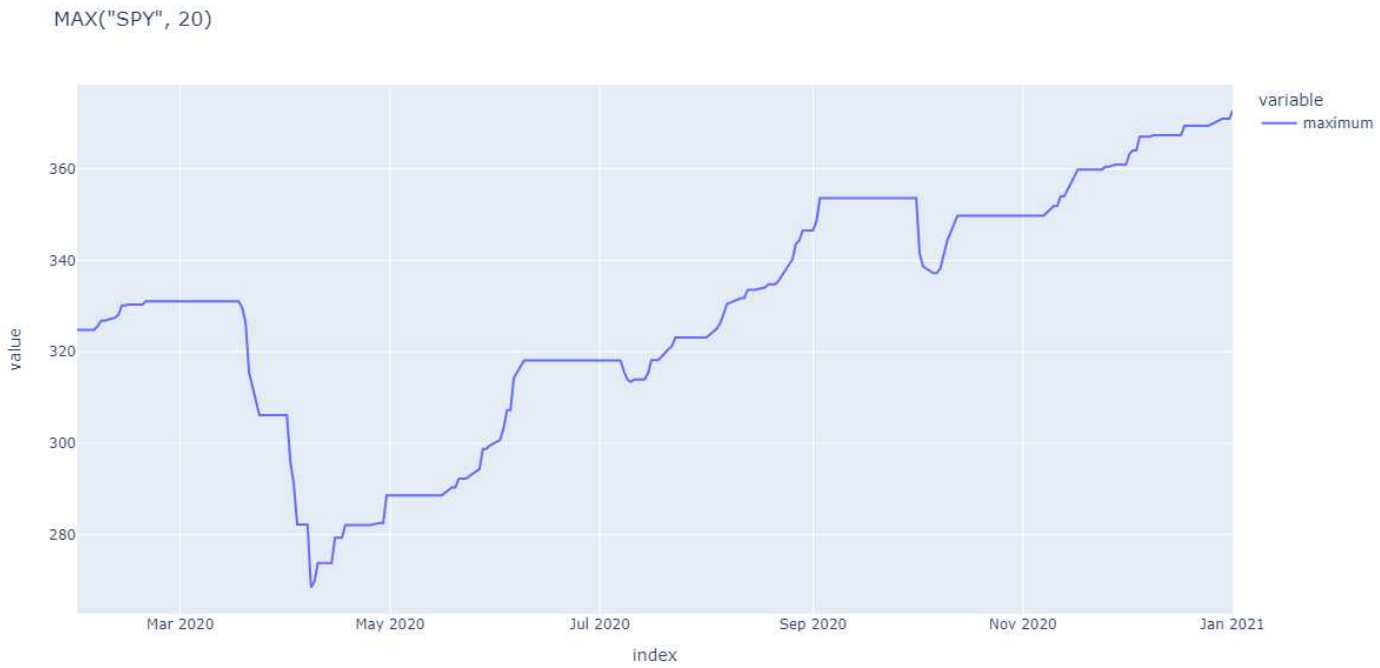
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class MaximumAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Maximum _maximum;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _maximum = new Maximum(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _maximum.Update(bar.EndTime, bar.Close);

        if (_maximum.IsReady)
        {
            // The current value of _maximum is represented by itself (_maximum)
            // or _maximum.Current.Value
            Plot("Maximum", "maximum", _maximum);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `Maximum` indicator properties:



Indicator History

To get the historical data of the `Maximum` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class MaximumAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Maximum _max;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _max = MAX(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_max, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_max, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_max, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

McClellan Oscillator

Introduction

The McClellan Oscillator is a market breadth indicator which was developed by Sherman and Marian McClellan. It is based on the difference between the number of advancing and declining periods.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MOSC Indicator

To create an automatic indicator for `McClellanOscillator` , call the `MOSC` helper method from the `QCAlgorithm` class. The `MOSC` method creates a `McClellanOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class McClellanOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private McClellanOscillator _mosc;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _mosc = MOSC([_symbol, _reference], 19, 39);
    }

    public override void OnData(Slice data)
    {

        if (_mosc.IsReady)
        {
            // The current value of _mosc is represented by itself (_mosc)
            // or _mosc.Current.Value
            Plot("McClellanOscillator", "mosc", _mosc);
            // Plot all properties of abands
            Plot("McClellanOscillator", "emafast", _mosc.EMAFast);
            Plot("McClellanOscillator", "emaslow", _mosc.EMASlow);
            Plot("McClellanOscillator", "addifference", _mosc.ADDifference);
        }
    }
}
```

To create an automatic indicator for `McClellanOscillator` using universe constituents, call the `MOSC` helper method from the `QCAlgorithm` class. The `MOSC` method creates a `McClellanOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In this case, you should call the helper method in the `OnSecuritiesChanged` method.

```C#
public class McClellanOscillatorAlgorithm : QCAlgorithm
{
    private Universe _universe;
    private McClellanOscillator _mosc;

    public override void Initialize()
    {
        SetStartDate(2024,1,1);
        SetEndDate(2024,12,31);
        UniverseSettings.Resolution = Resolution.Daily;
        UniverseSettings.Schedule.On(DateRules.MonthStart());
        _universe = AddUniverse(Universe.ETF("SPY"));
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        _mosc = MOSC(_universe.Selected);
    }

    public override void OnData(Slice data)
    {

        if (_mosc.IsReady)
        {
            // The current value of _mosc is represented by itself (_mosc)
            // or _mosc.Current.Value
            Plot("McClellanOscillator", "mosc", _mosc);
            // Plot all properties of abands
            Plot("McClellanOscillator", "emafast", _mosc.EMAFast);
            Plot("McClellanOscillator", "emaslow", _mosc.EMASlow);
            Plot("McClellanOscillator", "addifference", _mosc.ADDifference);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `McClellanOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

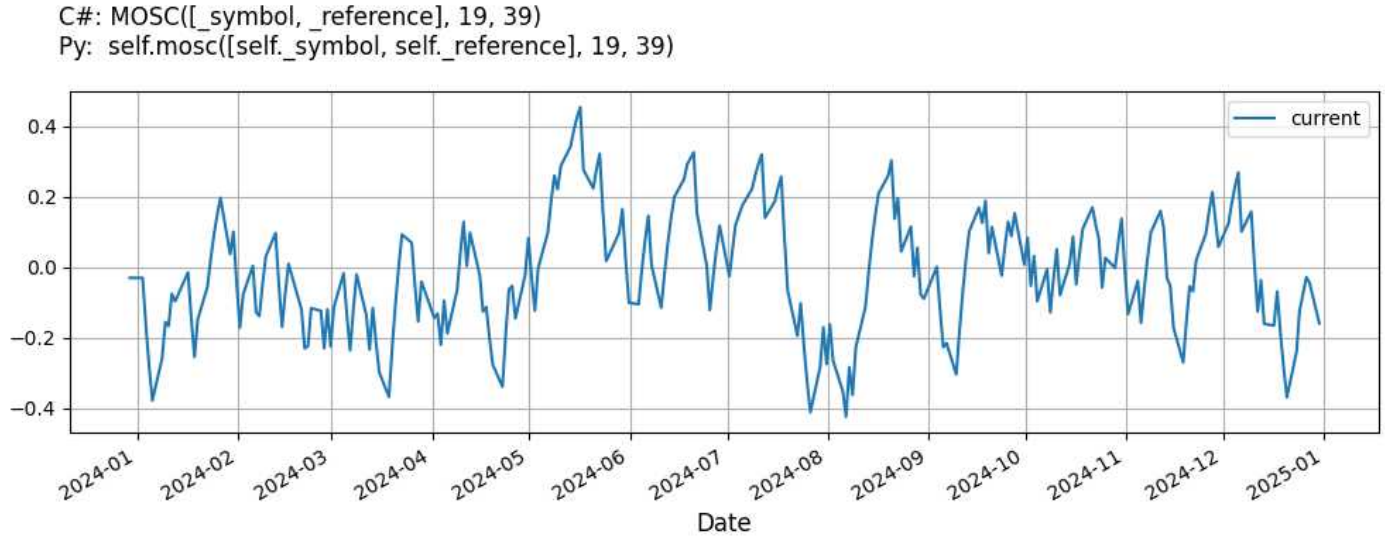
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class McClellanOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private McClellanOscillator _mcclellanoscillator;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _mcclellanoscillator = new McClellanOscillator(19, 39);
        _mcclellanoscillator.Add(_symbol);
        _mcclellanoscillator.Add(_reference);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _mcclellanoscillator.Update(bar);
        if (data.Bars.TryGetValue(_reference, out bar))
            _mcclellanoscillator.Update(bar);

        if (_mcclellanoscillator.IsReady)
        {
            // The current value of _mcclellanoscillator is represented by itself (_mcclellanoscillator)
            // or _mcclellanoscillator.Current.Value
            Plot("McClellanOscillator", "mcclellanoscillator", _mcclellanoscillator);
            // Plot all properties of abands
            Plot("McClellanOscillator", "emafast", _mcclellanoscillator.EMAFast);
            Plot("McClellanOscillator", "emaslow", _mcclellanoscillator.EMASlow);
            Plot("McClellanOscillator", "addifference", _mcclellanoscillator.ADDifference);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `McClellanOscillator` indicator properties:



Indicator History

To get the historical data of the `McClellanOscillator` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class McClellanOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private McClellanOscillator _mosc;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _mosc = MOSC([_symbol, _reference], 19, 39);

        var indicatorHistory = IndicatorHistory(_mosc, new[] { _symbol, _reference }, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_mosc, new[] { _symbol, _reference }, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_mosc, new[] { _symbol, _reference }, new DateTime(2024, 7,
1), new DateTime(2024, 7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var eMAFast = indicatorHistory.Select(x => ((dynamic)x).EMAFast).ToList();
        var eMASlow = indicatorHistory.Select(x => ((dynamic)x).EMASlow).ToList();
        var aDDifference = indicatorHistory.Select(x => ((dynamic)x).ADDifference).ToList();
    }
}
```

Supported Indicators

McClellan Summation Index

Introduction

The McClellan Summation Index (MSI) is a market breadth indicator that is based on the rolling average of difference between the number of advancing and declining issues on a stock exchange. It is generally considered as is a long-term version of the

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MSI Indicator

To create an automatic indicator for `McClellanSummationIndex` , call the `MSI` helper method from the `QCAlgorithm` class. The `MSI` method creates a `McClellanSummationIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

C#

```
public class McClellanSummationIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private McClellanSummationIndex _msi;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _msi = MSI([_symbol, _reference]);
    }

    public override void OnData(Slice data)
    {

        if (_msi.IsReady)
        {
            // The current value of _msi is represented by itself (_msi)
            // or _msi.Current.Value
            Plot("McClellanSummationIndex", "msi", _msi);
            // Plot all properties of abands
            Plot("McClellanSummationIndex", "mcclellanoscillator", _msi.McClellanOscillator);
        }
    }
}
```

To create an automatic indicator for `McClellanSummationIndex` using universe constituents, call the `MSI` helper method from the `QCAlgorithm` class. The `MSI` method creates a `McClellanSummationIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In this case, you should call the helper method in the `OnSecuritiesChanged` method.

```C#
public class McClellanSummationIndexAlgorithm : QCAlgorithm
{
    private Universe _universe;
    private McClellanSummationIndex _msi;

    public override void Initialize()
    {
        SetStartDate(2024,1,1);
        SetEndDate(2024,12,31);
        UniverseSettings.Resolution = Resolution.Daily;
        UniverseSettings.Schedule.On(DateRules.MonthStart());
        _universe = AddUniverse(Universe.ETF("SPY"));
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        _msi = MSI(_universe.Selected);
    }

    public override void OnData(Slice data)
    {

        if (_msi.IsReady)
        {
            // The current value of _msi is represented by itself (_msi)
            // or _msi.Current.Value
            Plot("McClellanSummationIndex", "msi", _msi);
            // Plot all properties of abands
            Plot("McClellanSummationIndex", "mcclellanoscillator", _msi.McClellanOscillator);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `McClellanSummationIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

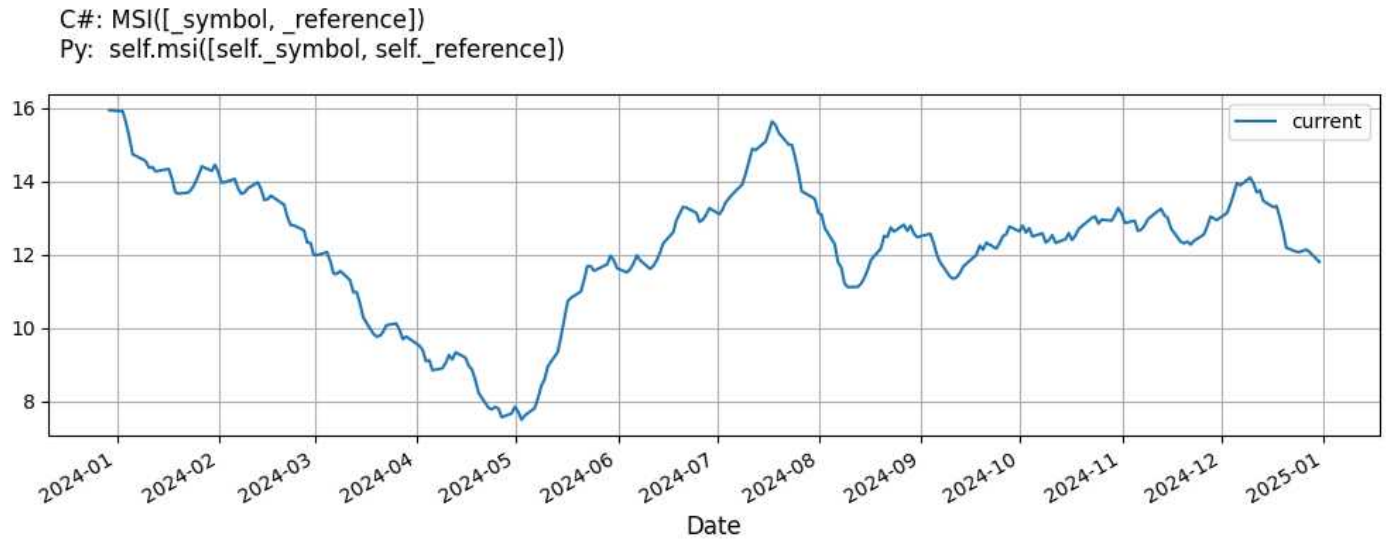
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class McClellanSummationIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private McClellanSummationIndex _mcclellansummationindex;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _mcclellansummationindex = new McClellanSummationIndex(19, 39);
        _mcclellansummationindex.Add(_symbol);
        _mcclellansummationindex.Add(_reference);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _mcclellansummationindex.Update(bar);
        if (data.Bars.TryGetValue(_reference, out bar))
            _mcclellansummationindex.Update(bar);

        if (_mcclellansummationindex.IsReady)
        {
            // The current value of _mcclellansummationindex is represented by itself (_mcclellansummationindex)
            // or _mcclellansummationindex.Current.Value
            Plot("McClellanSummationIndex", "mcclellansummationindex", _mcclellansummationindex);
            // Plot all properties of abands
            Plot("McClellanSummationIndex", "mcclellanoscillator", _mcclellansummationindex.McClellanOscillator);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `McClellanSummationIndex` indicator properties:



Indicator History

To get the historical data of the `McClellanSummationIndex` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class McClellanSummationIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private McClellanSummationIndex _msi;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _msi = MSI([_symbol, _reference]);

        var indicatorHistory = IndicatorHistory(_msi, new[] { _symbol, _reference }, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_msi, new[] { _symbol, _reference }, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_msi, new[] { _symbol, _reference }, new DateTime(2024, 7,
1), new DateTime(2024, 7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var mcClellanOscillator = indicatorHistory.Select(x => ((dynamic)x).McClellanOscillator).ToList();
    }
}
```

Supported Indicators

McGinley Dynamic

Introduction

This indicator represents the McGinley Dynamic (MGD) It is a type of moving average that was designed to track the market better than existing moving average indicators. It is a technical indicator that improves upon moving average lines by adjusting for shifts in market speed.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MGD Indicator

To create an automatic indicator for `McGinleyDynamic` , call the `MGD` helper method from the `QCAlgorithm` class. The `MGD` method creates a `McGinleyDynamic` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class McGinleyDynamicAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private McGinleyDynamic _mgd;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _mgd = MGD(_symbol, 10);
    }

    public override void OnData(Slice data)
    {

        if (_mgd.IsReady)
        {
            // The current value of _mgd is represented by itself (_mgd)
            // or _mgd.Current.Value
            Plot("McGinleyDynamic", "mgd", _mgd);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `McGinleyDynamic` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

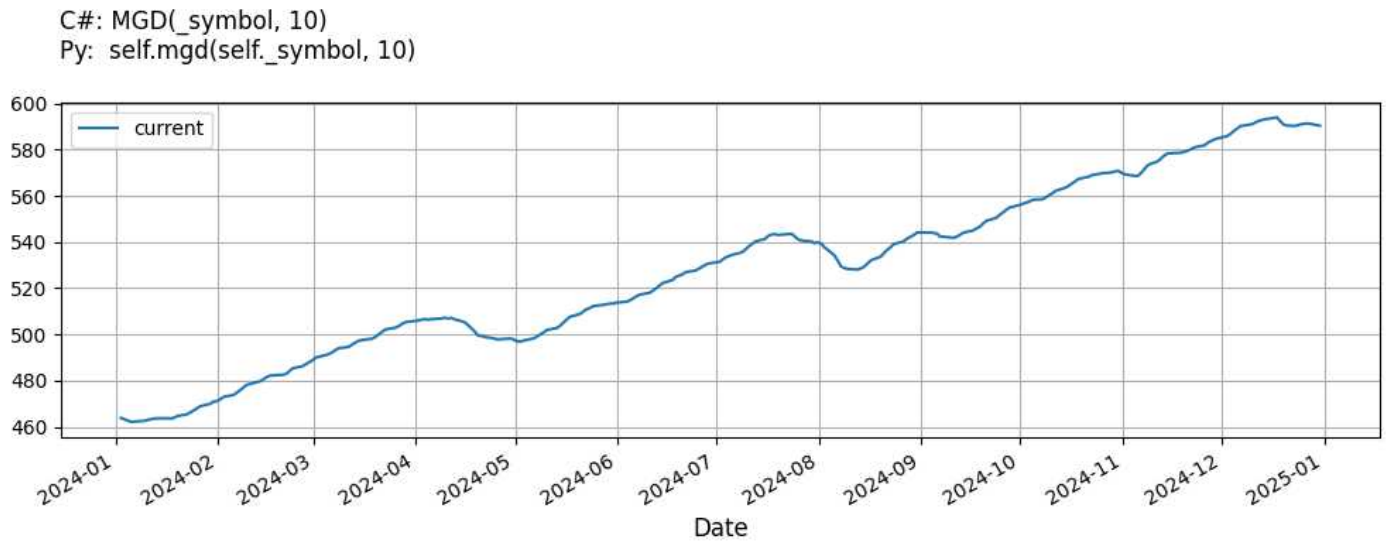
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class McGinleyDynamicAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private McGinleyDynamic _mcginleydynamic;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _mcginleydynamic = new McGinleyDynamic(10);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _mcginleydynamic.Update(bar.EndTime, bar.Close);

        if (_mcginleydynamic.IsReady)
        {
            // The current value of _mcginleydynamic is represented by itself (_mcginleydynamic)
            // or _mcginleydynamic.Current.Value
            Plot("McGinleyDynamic", "mcginleydynamic", _mcginleydynamic);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `McGinleyDynamic` indicator properties:



Indicator History

To get the historical data of the `McGinleyDynamic` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class McGinleyDynamicAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private McGinleyDynamic _mgd;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _mgd = MGD(_symbol, 10);

        var indicatorHistory = IndicatorHistory(_mgd, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_mgd, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_mgd, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Mean Absolute Deviation

Introduction

This indicator computes the n-period mean absolute deviation.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MAD Indicator

To create an automatic indicator for `MeanAbsoluteDeviation` , call the `MAD` helper method from the `QCAlgorithm` class. The `MAD` method creates a `MeanAbsoluteDeviation` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class MeanAbsoluteDeviationAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MeanAbsoluteDeviation _mad;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _mad = MAD(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_mad.IsReady)
        {
            // The current value of _mad is represented by itself (_mad)
            // or _mad.Current.Value
            Plot("MeanAbsoluteDeviation", "mad", _mad);
            // Plot all properties of abands
            Plot("MeanAbsoluteDeviation", "mean", _mad.Mean);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `MeanAbsoluteDeviation` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

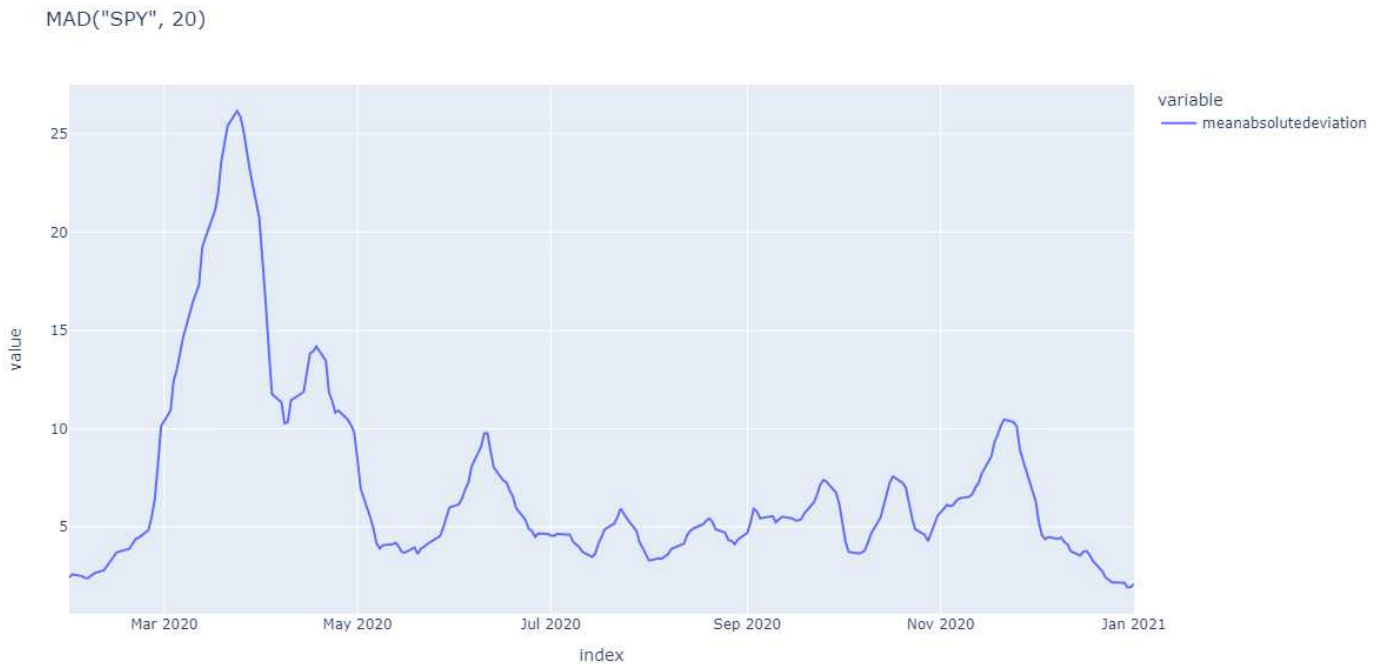
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class MeanAbsoluteDeviationAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MeanAbsoluteDeviation _meanabsolutedeviation;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _meanabsolutedeviation = new MeanAbsoluteDeviation(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _meanabsolutedeviation.Update(bar.EndTime, bar.Close);

        if (_meanabsolutedeviation.IsReady)
        {
            // The current value of _meanabsolutedeviation is represented by itself (_meanabsolutedeviation)
            // or _meanabsolutedeviation.Current.Value
            Plot("MeanAbsoluteDeviation", "meanabsolutedeviation", _meanabsolutedeviation);
            // Plot all properties of abands
            Plot("MeanAbsoluteDeviation", "mean", _meanabsolutedeviation.Mean);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `MeanAbsoluteDeviation` indicator properties:



Indicator History

To get the historical data of the `MeanAbsoluteDeviation` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class MeanAbsoluteDeviationAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MeanAbsoluteDeviation _mad;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _mad = MAD(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_mad, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_mad, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_mad, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var mean = indicatorHistory.Select(x => ((dynamic)x).Mean).ToList();
    }
}
```

Supported Indicators

Mesa Adaptive Moving Average

Introduction

Implements the Mesa Adaptive Moving Average (MAMA) indicator along with the following FAMA (Following Adaptive Moving Average) as a secondary indicator. The MAMA adjusts its smoothing factor based on the market's volatility, making it more adaptive than a simple moving average.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MAMA Indicator

To create an automatic indicator for `MesaAdaptiveMovingAverage` , call the `MAMA` helper method from the `QCAlgorithm` class. The `MAMA` method creates a `MesaAdaptiveMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class MesaAdaptiveMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MesaAdaptiveMovingAverage _mama;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _mama = MAMA(_symbol, 0.5m, 0.05m);
    }

    public override void OnData(Slice data)
    {

        if (_mama.IsReady)
        {
            // The current value of _mama is represented by itself (_mama)
            // or _mama.Current.Value
            Plot("MesaAdaptiveMovingAverage", "mama", _mama);
            // Plot all properties of abands
            Plot("MesaAdaptiveMovingAverage", "fama", _mama.Fama);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `MesaAdaptiveMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

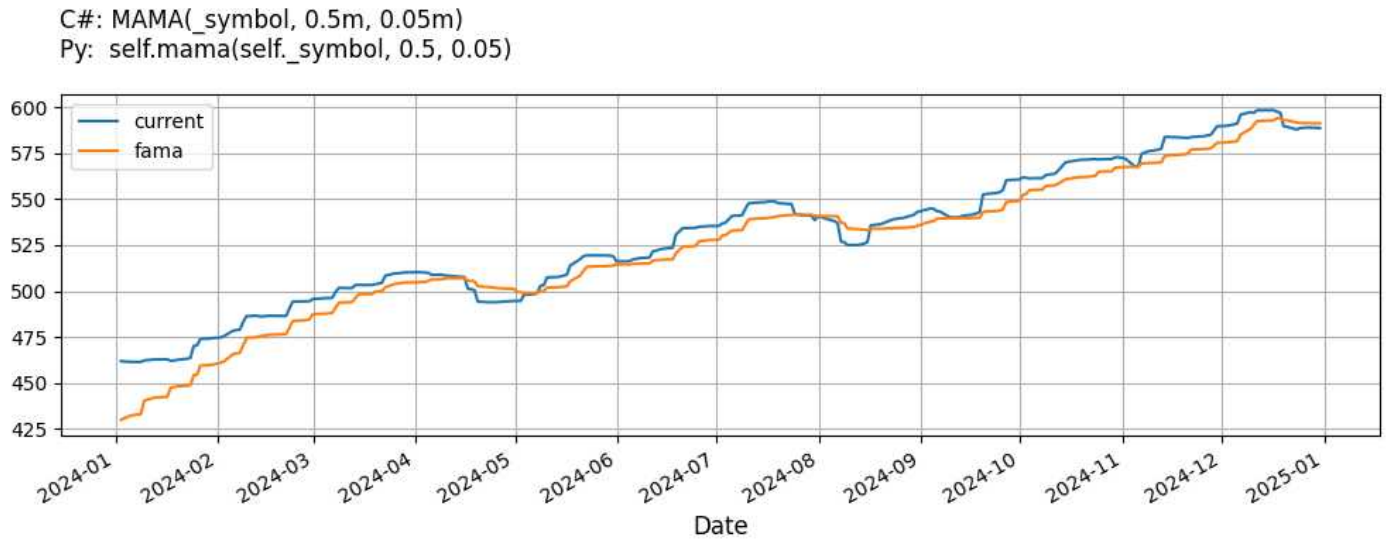
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class MesaAdaptiveMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MesaAdaptiveMovingAverage _mesaadaptivemovingaverage;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _mesaadaptivemovingaverage = new MesaAdaptiveMovingAverage(0.5m, 0.05m);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _mesaadaptivemovingaverage.Update(bar);

        if (_mesaadaptivemovingaverage.IsReady)
        {
            // The current value of _mesaadaptivemovingaverage is represented by itself (_mesaadaptivemovingaverage)
            // or _mesaadaptivemovingaverage.Current.Value
            Plot("MesaAdaptiveMovingAverage", "mesaadaptivemovingaverage", _mesaadaptivemovingaverage);
            // Plot all properties of abands
            Plot("MesaAdaptiveMovingAverage", "fama", _mesaadaptivemovingaverage.Fama);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `MesaAdaptiveMovingAverage` indicator properties:



Indicator History

To get the historical data of the `MesaAdaptiveMovingAverage` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class MesaAdaptiveMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MesaAdaptiveMovingAverage _mama;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _mama = MAMA(_symbol, 0.5m, 0.05m);

        var indicatorHistory = IndicatorHistory(_mama, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_mama, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_mama, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var fama = indicatorHistory.Select(x => ((dynamic)x).Fama).ToList();
    }
}
```

Supported Indicators

Mid Point

Introduction

This indicator computes the MidPoint (MIDPOINT) The MidPoint is calculated using the following formula: MIDPOINT = (Highest Value + Lowest Value) / 2

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MIDPOINT Indicator

To create an automatic indicator for `MidPoint` , call the `MIDPOINT` helper method from the `QCAlgorithm` class. The `MIDPOINT` method creates a `MidPoint` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class MidPointAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MidPoint _midpoint;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _midpoint = MIDPOINT(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_midpoint.IsReady)
        {
            // The current value of _midpoint is represented by itself (_midpoint)
            // or _midpoint.Current.Value
            Plot("MidPoint", "midpoint", _midpoint);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `MidPoint` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

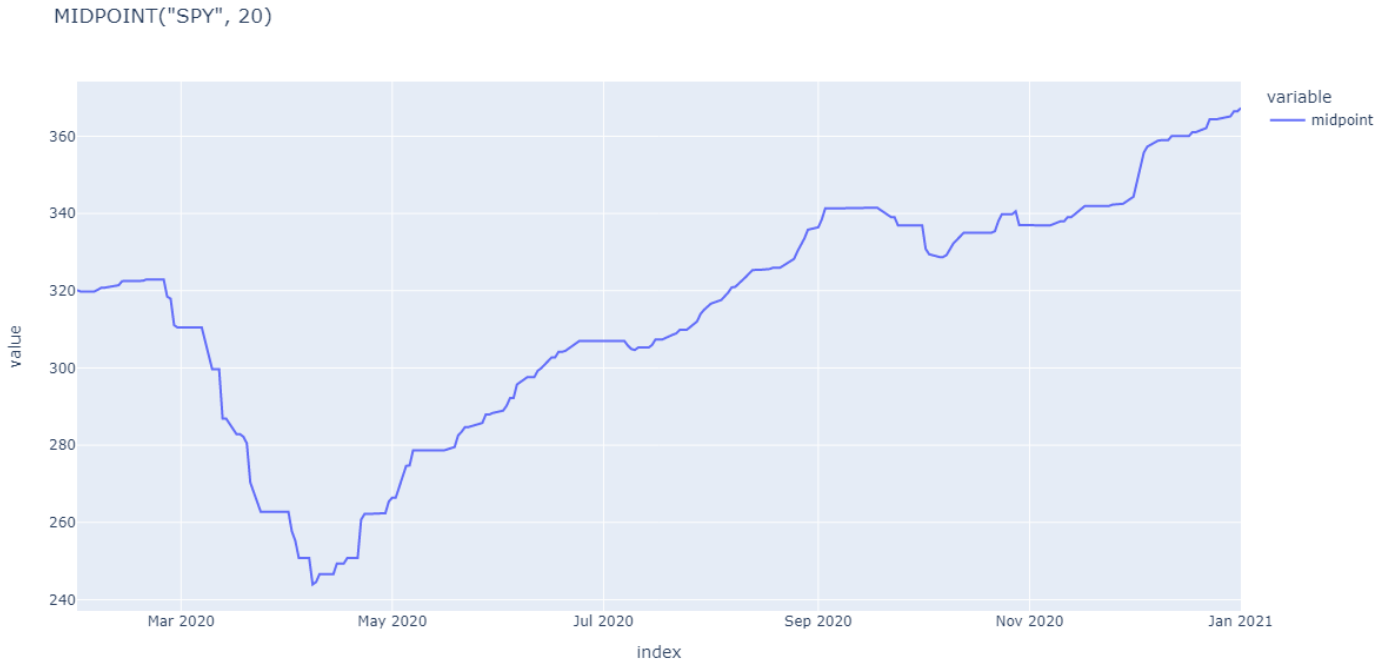
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class MidPointAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MidPoint _midpoint;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _midpoint = new MidPoint(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _midpoint.Update(bar.EndTime, bar.Close);

        if (_midpoint.IsReady)
        {
            // The current value of _midpoint is represented by itself (_midpoint)
            // or _midpoint.Current.Value
            Plot("MidPoint", "midpoint", _midpoint);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `MidPoint` indicator properties:



Indicator History

To get the historical data of the `MidPoint` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class MidPointAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MidPoint _midpoint;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _midpoint = MIDPOINT(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_midpoint, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_midpoint, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_midpoint, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Mid Price

Introduction

This indicator computes the MidPrice (MIDPRICE). The MidPrice is calculated using the following formula: MIDPRICE = (Highest High + Lowest Low) / 2

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MIDPRICE Indicator

To create an automatic indicator for `MidPrice` , call the `MIDPRICE` helper method from the `QCAlgorithm` class. The `MIDPRICE` method creates a `MidPrice` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class MidPriceAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MidPrice _midprice;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _midprice = MIDPRICE(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_midprice.IsReady)
        {
            // The current value of _midprice is represented by itself (_midprice)
            // or _midprice.Current.Value
            Plot("MidPrice", "midprice", _midprice);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `MidPrice` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

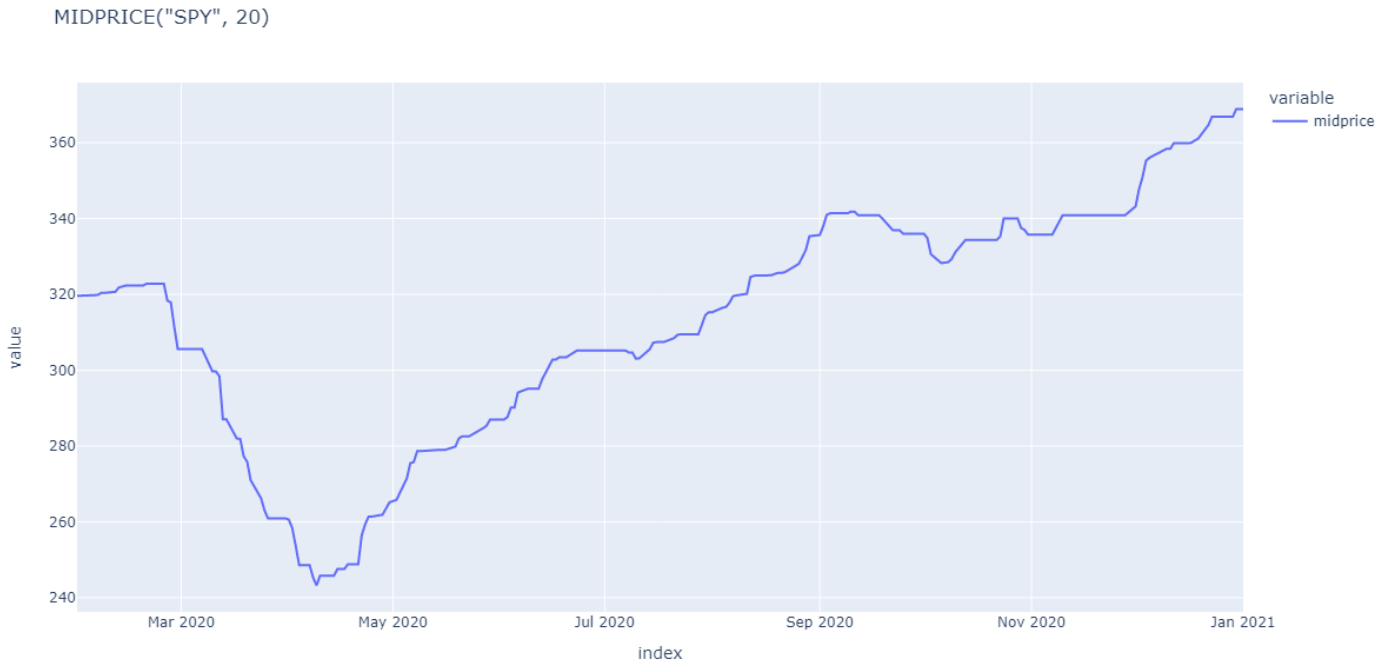
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class MidPriceAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MidPrice _midprice;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _midprice = new MidPrice(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _midprice.Update(bar);

        if (_midprice.IsReady)
        {
            // The current value of _midprice is represented by itself (_midprice)
            // or _midprice.Current.Value
            Plot("MidPrice", "midprice", _midprice);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `MidPrice` indicator properties:



Indicator History

To get the historical data of the `MidPrice` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class MidPriceAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MidPrice _midprice;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _midprice = MIDPRICE(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_midprice, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_midprice, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_midprice, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Minimum

Introduction

This indicator represents an indicator capable of tracking the minimum value and how many periods ago it occurred

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MIN Indicator

To create an automatic indicator for `Minimum` , call the `MIN` helper method from the `QCAlgorithm` class. The `MIN` method creates a `Minimum` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class MinimumAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Minimum _min;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _min = MIN(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_min.IsReady)
        {
            // The current value of _min is represented by itself (_min)
            // or _min.Current.Value
            Plot("Minimum", "min", _min);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `Minimum` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

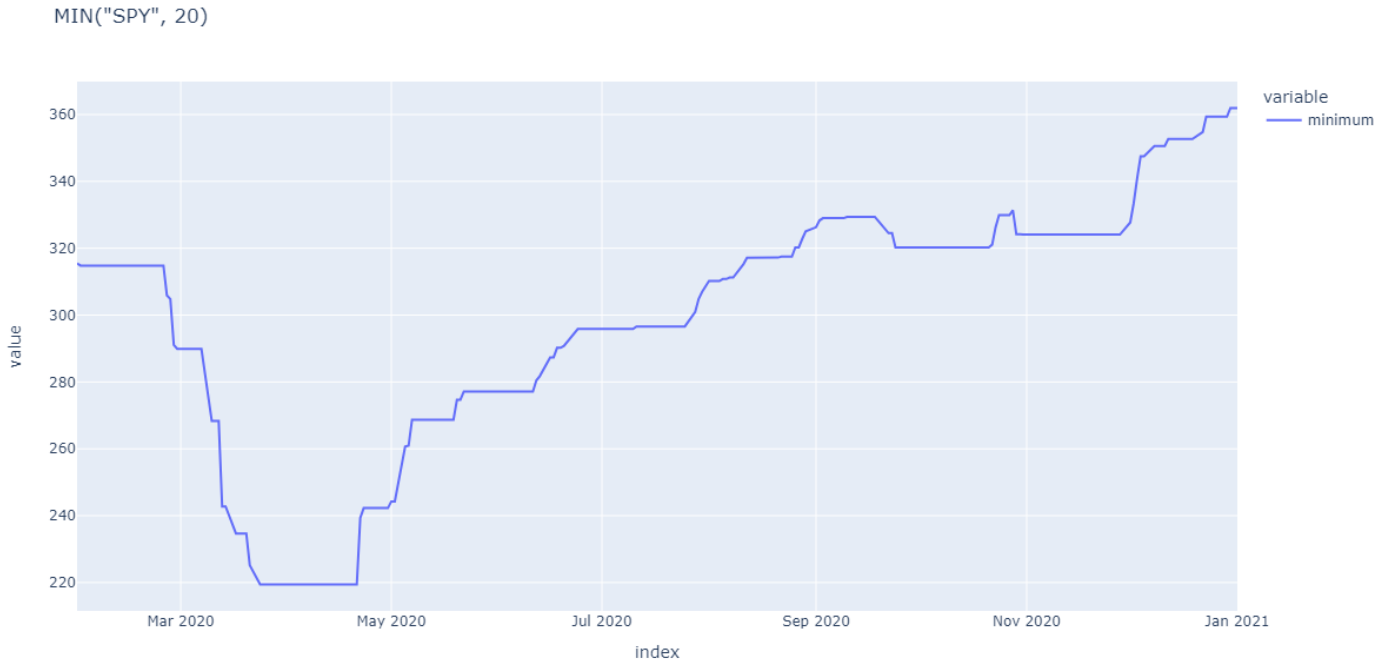
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class MinimumAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Minimum _minimum;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _minimum = new Minimum(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _minimum.Update(bar.EndTime, bar.Close);

        if (_minimum.IsReady)
        {
            // The current value of _minimum is represented by itself (_minimum)
            // or _minimum.Current.Value
            Plot("Minimum", "minimum", _minimum);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `Minimum` indicator properties:



Indicator History

To get the historical data of the `Minimum` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class MinimumAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Minimum _min;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _min = MIN(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_min, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_min, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_min, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Momentum

Introduction

This indicator computes the n-period change in a value using the following: value_0 - value_n

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MOM Indicator

To create an automatic indicator for `Momentum` , call the `MOM` helper method from the `QCAlgorithm` class. The `MOM` method creates a `Momentum` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class MomentumAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Momentum _mom;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _mom = MOM(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_mom.IsReady)
        {
            // The current value of _mom is represented by itself (_mom)
            // or _mom.Current.Value
            Plot("Momentum", "mom", _mom);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `Momentum` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class MomentumAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Momentum _momentum;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _momentum = new Momentum(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _momentum.Update(bar.EndTime, bar.Close);

        if (_momentum.IsReady)
        {
            // The current value of _momentum is represented by itself (_momentum)
            // or _momentum.Current.Value
            Plot("Momentum", "momentum", _momentum);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `Momentum` indicator properties:

Indicator History

To get the historical data of the `Momentum` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class MomentumAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Momentum _mom;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _mom = MOM(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_mom, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_mom, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_mom, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Momentum Percent

Introduction

This indicator computes the n-period percentage rate of change in a value using the following: 100 * (value_0 - value_n) / value_n This indicator yields the same results of RateOfChangePercent

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MOMP Indicator

To create an automatic indicator for `MomentumPercent` , call the `MOMP` helper method from the `QCAlgorithm` class. The `MOMP` method creates a `MomentumPercent` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class MomentumPercentAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MomentumPercent _momp;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _momp = MOMP(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_momp.IsReady)
        {
            // The current value of _momp is represented by itself (_momp)
            // or _momp.Current.Value
            Plot("MomentumPercent", "momp", _momp);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `MomentumPercent` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

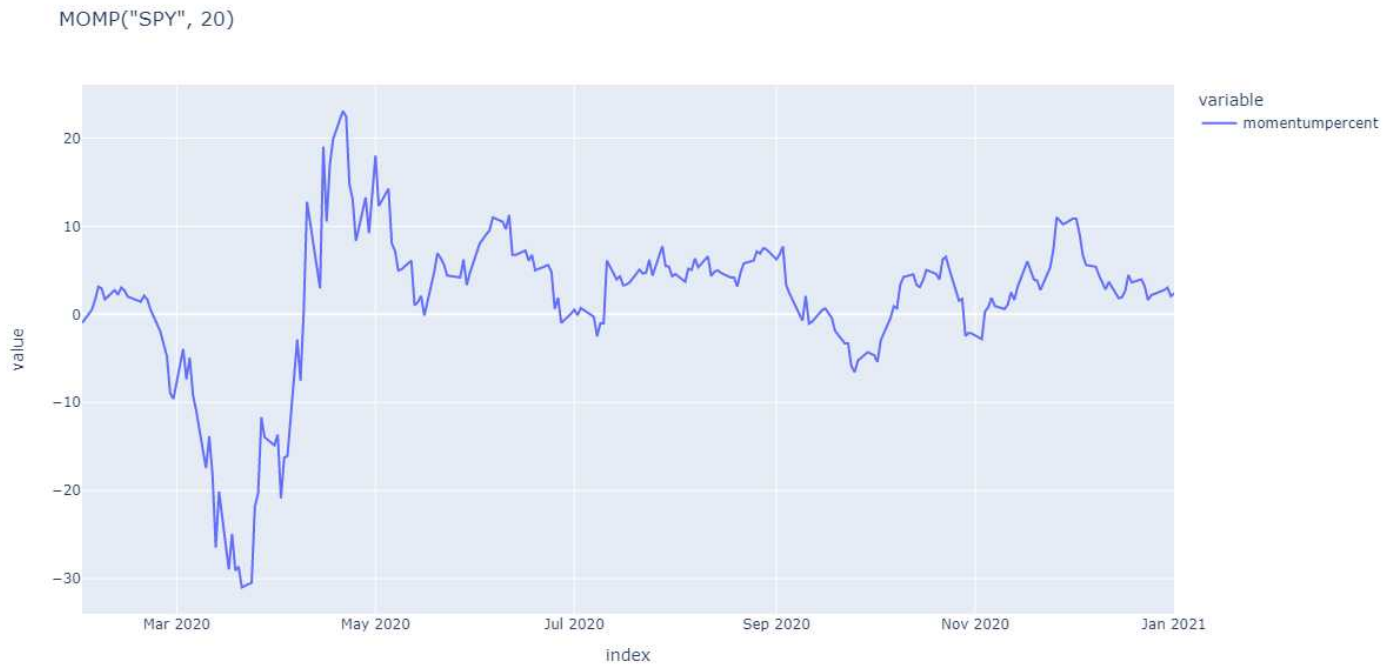
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class MomentumPercentAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MomentumPercent _momentumpercent;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _momentumpercent = new MomentumPercent(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _momentumpercent.Update(bar.EndTime, bar.Close);

        if (_momentumpercent.IsReady)
        {
            // The current value of _momentumpercent is represented by itself (_momentumpercent)
            // or _momentumpercent.Current.Value
            Plot("MomentumPercent", "momentumpercent", _momentumpercent);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `MomentumPercent` indicator properties:



Indicator History

To get the historical data of the `MomentumPercent` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class MomentumPercentAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MomentumPercent _momp;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _momp = MOMP(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_momp, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_momp, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_momp, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Momersion

Introduction

Oscillator indicator that measures momentum and mean-reversion over a specified period n. [source] Harris, Michael. "Momersion Indicator." Price Action Lab., 13 Aug. 2015. Web.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MOMERSION Indicator

To create an automatic indicator for `Momersion` , call the `MOMERSION` helper method from the `QCAlgorithm` class. The `MOMERSION` method creates a `Momersion` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class MomersionAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Momersion _momersion;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _momersion = MOMERSION(_symbol, 10, 20);
    }

    public override void OnData(Slice data)
    {

        if (_momersion.IsReady)
        {
            // The current value of _momersion is represented by itself (_momersion)
            // or _momersion.Current.Value
            Plot("Momersion", "momersion", _momersion);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `Momersion` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

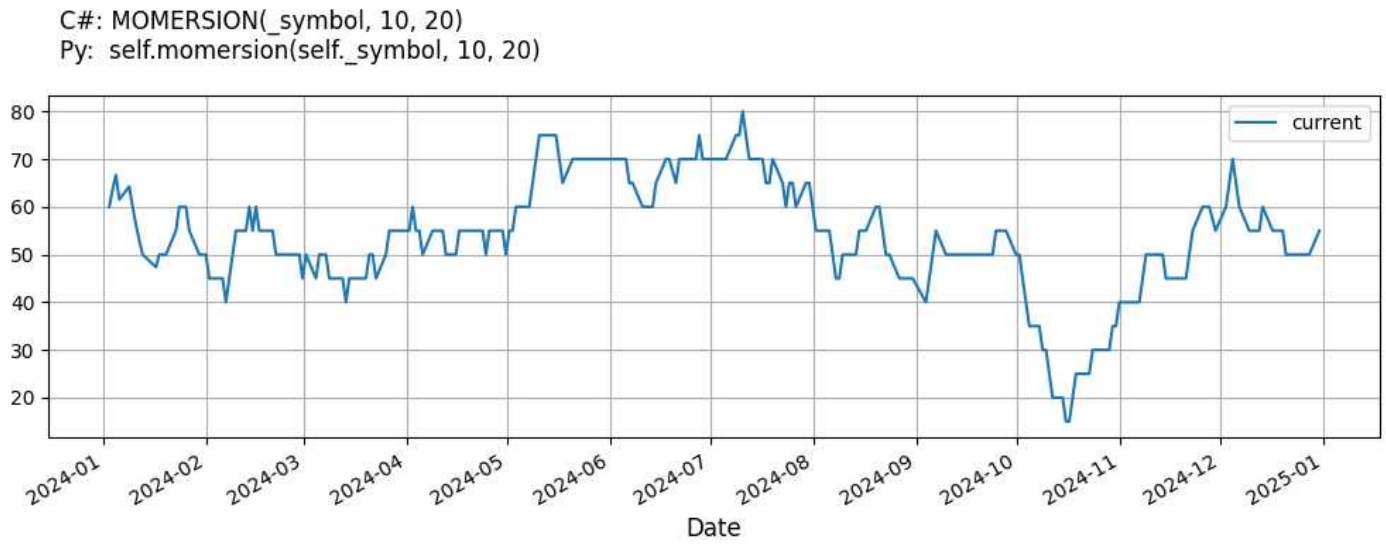
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class MomersionAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Momersion _momersion;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _momersion = new Momersion(10, 20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _momersion.Update(bar.EndTime, bar.Close);

        if (_momersion.IsReady)
        {
            // The current value of _momersion is represented by itself (_momersion)
            // or _momersion.Current.Value
            Plot("Momersion", "momersion", _momersion);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `Momersion` indicator properties:



Indicator History

To get the historical data of the `Momersion` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class MomersionAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Momersion _momersion;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _momersion = MOMERSION(_symbol, 10, 20);

        var indicatorHistory = IndicatorHistory(_momersion, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_momersion, _symbol, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_momersion, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Money Flow Index

Introduction

The Money Flow Index (MFI) is an oscillator that uses both price and volume to measure buying and selling pressure Typical Price = (High + Low + Close)/3 Money Flow = Typical Price x Volume Positive Money Flow = Sum of the money flows of all days where the typical price is greater than the previous day's typical price Negative Money Flow = Sum of the money flows of all days where the typical price is less than the previous day's typical price Money Flow Ratio = (14-period Positive Money Flow)/(14-period Negative Money Flow) Money Flow Index = 100 x Positive Money Flow / (Positive Money Flow + Negative Money Flow)

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MFI Indicator

To create an automatic indicator for `MoneyFlowIndex` , call the `MFI` helper method from the `QCAlgorithm` class. The `MFI` method creates a `MoneyFlowIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

C#

```
public class MoneyFlowIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MoneyFlowIndex _mfi;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _mfi = MFI(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_mfi.IsReady)
        {
            // The current value of _mfi is represented by itself (_mfi)
            // or _mfi.Current.Value
            Plot("MoneyFlowIndex", "mfi", _mfi);
            // Plot all properties of abands
            Plot("MoneyFlowIndex", "positivemoneyflow", _mfi.PositiveMoneyFlow);
            Plot("MoneyFlowIndex", "negativemoneyflow", _mfi.NegativeMoneyFlow);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `MoneyFlowIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

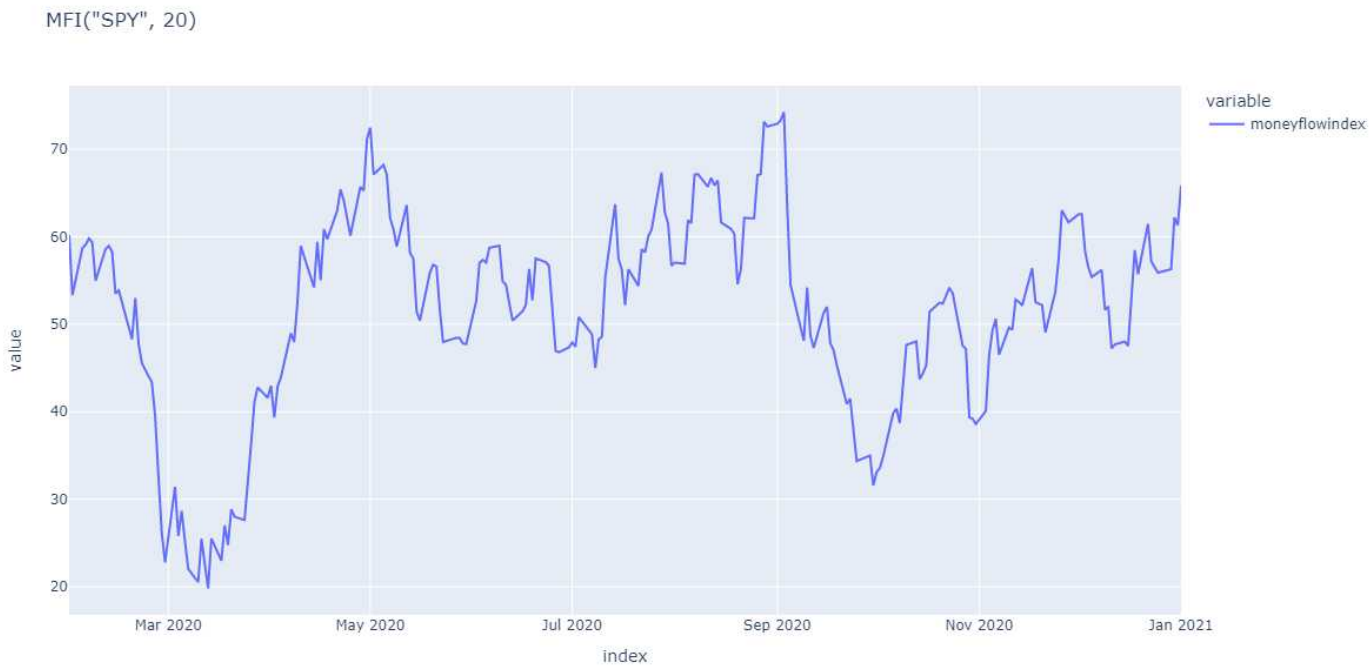
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class MoneyFlowIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MoneyFlowIndex _moneyflowindex;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _moneyflowindex = new MoneyFlowIndex(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _moneyflowindex.Update(bar);

        if (_moneyflowindex.IsReady)
        {
            // The current value of _moneyflowindex is represented by itself (_moneyflowindex)
            // or _moneyflowindex.Current.Value
            Plot("MoneyFlowIndex", "moneyflowindex", _moneyflowindex);
            // Plot all properties of abands
            Plot("MoneyFlowIndex", "positivemoneyflow", _moneyflowindex.PositiveMoneyFlow);
            Plot("MoneyFlowIndex", "negativemoneyflow", _moneyflowindex.NegativeMoneyFlow);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `MoneyFlowIndex` indicator properties:



Indicator History

To get the historical data of the `MoneyFlowIndex` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class MoneyFlowIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MoneyFlowIndex _mfi;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _mfi = MFI(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_mfi, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_mfi, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_mfi, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var positiveMoneyFlow = indicatorHistory.Select(x => ((dynamic)x).PositiveMoneyFlow).ToList();
        var negativeMoneyFlow = indicatorHistory.Select(x => ((dynamic)x).NegativeMoneyFlow).ToList();
    }
}
```

Supported Indicators

Moving Average Convergence Divergence

Introduction

This indicator creates two moving averages defined on a base indicator and produces the difference between the fast and slow averages.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using MACD Indicator

To create an automatic indicator for `MovingAverageConvergenceDivergence` , call the `MACD` helper method from the `QCAlgorithm` class. The `MACD` method creates a `MovingAverageConvergenceDivergence` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class MovingAverageConvergenceDivergenceAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MovingAverageConvergenceDivergence _macd;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _macd = MACD(_symbol, 12, 26, 9, MovingAverageType.Exponential);
    }

    public override void OnData(Slice data)
    {

        if (_macd.IsReady)
        {
            // The current value of _macd is represented by itself (_macd)
            // or _macd.Current.Value
            Plot("MovingAverageConvergenceDivergence", "macd", _macd);
            // Plot all properties of abands
            Plot("MovingAverageConvergenceDivergence", "fast", _macd.Fast);
            Plot("MovingAverageConvergenceDivergence", "slow", _macd.Slow);
            Plot("MovingAverageConvergenceDivergence", "signal", _macd.Signal);
            Plot("MovingAverageConvergenceDivergence", "histogram", _macd.Histogram);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `MovingAverageConvergenceDivergence` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

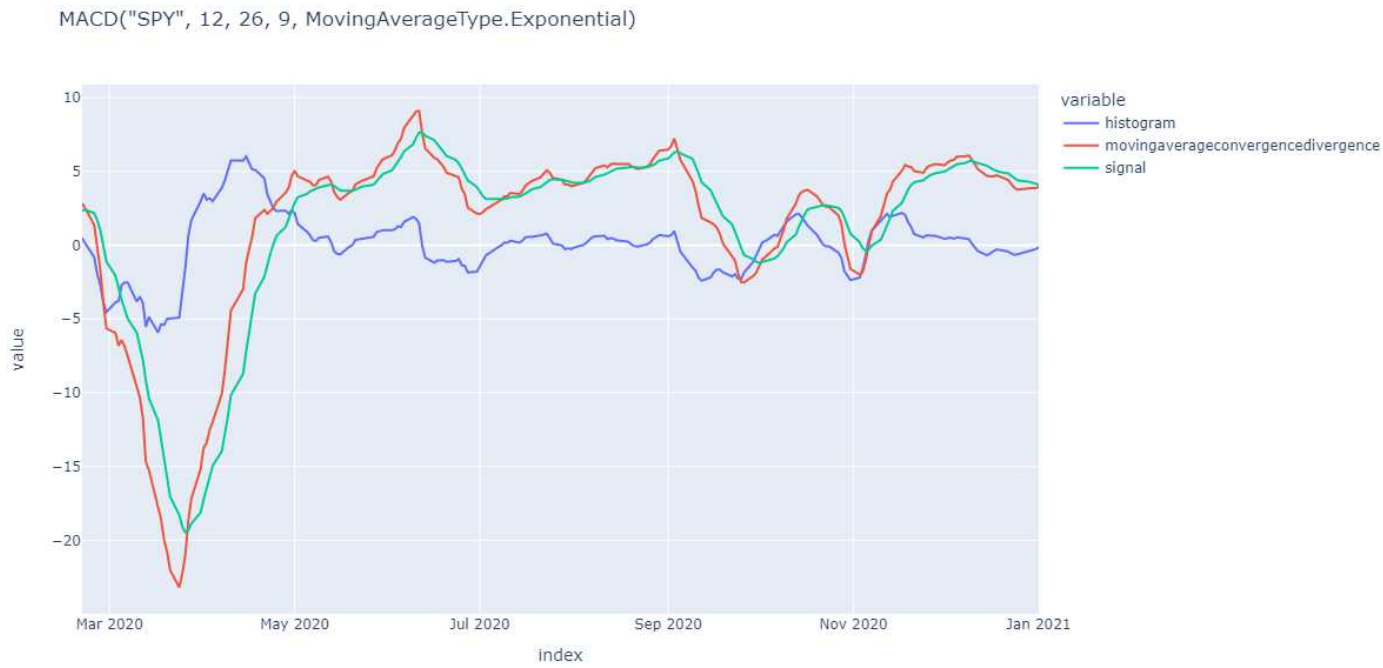
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class MovingAverageConvergenceDivergenceAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MovingAverageConvergenceDivergence _movingaverageconvergencedivergence;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _movingaverageconvergencedivergence = new MovingAverageConvergenceDivergence(12, 26, 9,
MovingAverageType.Exponential);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _movingaverageconvergencedivergence.Update(bar.EndTime, bar.Close);

        if (_movingaverageconvergencedivergence.IsReady)
        {
            // The current value of _movingaverageconvergencedivergence is represented by itself
(_movingaverageconvergencedivergence)
            // or _movingaverageconvergencedivergence.Current.Value
            Plot("MovingAverageConvergenceDivergence", "movingaverageconvergencedivergence",
_movingaverageconvergencedivergence);
            // Plot all properties of abands
            Plot("MovingAverageConvergenceDivergence", "fast", _movingaverageconvergencedivergence.Fast);
            Plot("MovingAverageConvergenceDivergence", "slow", _movingaverageconvergencedivergence.Slow);
            Plot("MovingAverageConvergenceDivergence", "signal", _movingaverageconvergencedivergence.Signal);
            Plot("MovingAverageConvergenceDivergence", "histogram", _movingaverageconvergencedivergence.Histogram);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `MovingAverageConvergenceDivergence` indicator properties:



Indicator History

To get the historical data of the `MovingAverageConvergenceDivergence` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular

history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class MovingAverageConvergenceDivergenceAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private MovingAverageConvergenceDivergence _macd;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _macd = MACD(_symbol, 12, 26, 9, MovingAverageType.Exponential);

        var indicatorHistory = IndicatorHistory(_macd, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_macd, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_macd, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var fast = indicatorHistory.Select(x => ((dynamic)x).Fast).ToList();
        var slow = indicatorHistory.Select(x => ((dynamic)x).Slow).ToList();
        var signal = indicatorHistory.Select(x => ((dynamic)x).Signal).ToList();
        var histogram = indicatorHistory.Select(x => ((dynamic)x).Histogram).ToList();
    }
}
```

Supported Indicators

New Highs New Lows

Introduction

The NewHighsNewLows is a New Highs / New Lows indicator that accepts IBaseDataBar data as its input. This type is more of a shim/typedef to reduce the need to refer to things as NewHighsNewLows<IBaseDataBar>

To view the implementation of this indicator, see the LEAN GitHub repository .

Using NHNL Indicator

To create an automatic indicator for `NewHighsNewLows` , call the `NHNL` helper method from the `QCAlgorithm` class. The `NHNL` method creates a `NewHighsNewLows` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

C#

```
public class NewHighsNewLowsAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private NewHighsNewLows _nhnl;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _nhnl = NHNL([_symbol, _reference]);
    }

    public override void OnData(Slice data)
    {

        if (_nhnl.IsReady)
        {
            // The current value of _nhnl is represented by itself (_nhnl)
            // or _nhnl.Current.Value
            Plot("NewHighsNewLows", "nhnl", _nhnl);
            // Plot all properties of abands
            Plot("NewHighsNewLows", "difference", _nhnl.Difference);
            Plot("NewHighsNewLows", "ratio", _nhnl.Ratio);
        }
    }
}
```

To create an automatic indicator for `NewHighsNewLows` using universe constituents, call the `NHNL` helper method from the `QCAlgorithm` class. The `NHNL` method creates a `NewHighsNewLows` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In this case, you should call the helper method in the `OnSecuritiesChanged` method.

```C#
public class NewHighsNewLowsAlgorithm : QCAlgorithm
{
    private Universe _universe;
    private NewHighsNewLows _nhnl;

    public override void Initialize()
    {
        SetStartDate(2024,1,1);
        SetEndDate(2024,12,31);
        UniverseSettings.Resolution = Resolution.Daily;
        UniverseSettings.Schedule.On(DateRules.MonthStart());
        _universe = AddUniverse(Universe.ETF("SPY"));
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        _nhnl = NHNL(_universe.Selected);
    }

    public override void OnData(Slice data)
    {

        if (_nhnl.IsReady)
        {
            // The current value of _nhnl is represented by itself (_nhnl)
            // or _nhnl.Current.Value
            Plot("NewHighsNewLows", "nhnl", _nhnl);
            // Plot all properties of abands
            Plot("NewHighsNewLows", "difference", _nhnl.Difference);
            Plot("NewHighsNewLows", "ratio", _nhnl.Ratio);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `NewHighsNewLows` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class NewHighsNewLowsAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private NewHighsNewLows _newhighsnewlows;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _newhighsnewlows = new NewHighsNewLows(19, 39);
        _newhighsnewlows.Add(_symbol);
        _newhighsnewlows.Add(_reference);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _newhighsnewlows.Update(bar);
        if (data.Bars.TryGetValue(_reference, out bar))
            _newhighsnewlows.Update(bar);

        if (_newhighsnewlows.IsReady)
        {
            // The current value of _newhighsnewlows is represented by itself (_newhighsnewlows)
            // or _newhighsnewlows.Current.Value
            Plot("NewHighsNewLows", "newhighsnewlows", _newhighsnewlows);
            // Plot all properties of abands
            Plot("NewHighsNewLows", "difference", _newhighsnewlows.Difference);
            Plot("NewHighsNewLows", "ratio", _newhighsnewlows.Ratio);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `NewHighsNewLows` indicator properties:

Indicator History

To get the historical data of the `NewHighsNewLows` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the

`IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class NewHighsNewLowsAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private NewHighsNewLows _nhnl;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _nhnl = NHNL([_symbol, _reference]);

        var indicatorHistory = IndicatorHistory(_nhnl, new[] { _symbol, _reference }, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_nhnl, new[] { _symbol, _reference }, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_nhnl, new[] { _symbol, _reference }, new DateTime(2024, 7,
1), new DateTime(2024, 7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var difference = indicatorHistory.Select(x => ((dynamic)x).Difference).ToList();
        var ratio = indicatorHistory.Select(x => ((dynamic)x).Ratio).ToList();
    }
}
```

Supported Indicators

New Highs New Lows Volume

Introduction

The New Highs - New Lows Volume Ratio is a Breadth indicator calculated as ratio of summary volume of stocks reaching new high to summary volume of stocks reaching new low compared to high and low values in defined time period.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using NHNLV Indicator

To create an automatic indicator for `NewHighsNewLowsVolume` , call the `NHNLV` helper method from the `QCAlgorithm` class. The `NHNLV` method creates a `NewHighsNewLowsVolume` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class NewHighsNewLowsVolumeAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private NewHighsNewLowsVolume _nhnlv;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _nhnlv = NHNLV([_symbol, _reference]);
    }

    public override void OnData(Slice data)
    {

        if (_nhnlv.IsReady)
        {
            // The current value of _nhnlv is represented by itself (_nhnlv)
            // or _nhnlv.Current.Value
            Plot("NewHighsNewLowsVolume", "nhnlv", _nhnlv);
            // Plot all properties of abands
            Plot("NewHighsNewLowsVolume", "volumeratio", _nhnlv.VolumeRatio);
            Plot("NewHighsNewLowsVolume", "difference", _nhnlv.Difference);
            Plot("NewHighsNewLowsVolume", "ratio", _nhnlv.Ratio);
        }
    }
}
```

To create an automatic indicator for `NewHighsNewLowsVolume` using universe constituents, call the `NHNLV` helper method from the `QCAlgorithm` class. The `NHNLV` method creates a `NewHighsNewLowsVolume` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In this case, you should call the helper method in the `OnSecuritiesChanged` method.

```C#
public class NewHighsNewLowsVolumeAlgorithm : QCAlgorithm
{
    private Universe _universe;
    private NewHighsNewLowsVolume _nhnlv;

    public override void Initialize()
    {
        SetStartDate(2024,1,1);
        SetEndDate(2024,12,31);
        UniverseSettings.Resolution = Resolution.Daily;
        UniverseSettings.Schedule.On(DateRules.MonthStart());
        _universe = AddUniverse(Universe.ETF("SPY"));
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        _nhnlv = NHNLV(_universe.Selected);
    }

    public override void OnData(Slice data)
    {

        if (_nhnlv.IsReady)
        {
            // The current value of _nhnlv is represented by itself (_nhnlv)
            // or _nhnlv.Current.Value
            Plot("NewHighsNewLowsVolume", "nhnlv", _nhnlv);
            // Plot all properties of abands
            Plot("NewHighsNewLowsVolume", "volumeratio", _nhnlv.VolumeRatio);
            Plot("NewHighsNewLowsVolume", "difference", _nhnlv.Difference);
            Plot("NewHighsNewLowsVolume", "ratio", _nhnlv.Ratio);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `NewHighsNewLowsVolume` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class NewHighsNewLowsVolumeAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private NewHighsNewLowsVolume _newhighsnewlowsvolume;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _newhighsnewlowsvolume = new NewHighsNewLowsVolume(19, 39);
        _newhighsnewlowsvolume.Add(_symbol);
        _newhighsnewlowsvolume.Add(_reference);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _newhighsnewlowsvolume.Update(bar);
        if (data.Bars.TryGetValue(_reference, out bar))
            _newhighsnewlowsvolume.Update(bar);

        if (_newhighsnewlowsvolume.IsReady)
        {
            // The current value of _newhighsnewlowsvolume is represented by itself (_newhighsnewlowsvolume)
            // or _newhighsnewlowsvolume.Current.Value
            Plot("NewHighsNewLowsVolume", "newhighsnewlowsvolume", _newhighsnewlowsvolume);
            // Plot all properties of abands
            Plot("NewHighsNewLowsVolume", "volumeratio", _newhighsnewlowsvolume.VolumeRatio);
            Plot("NewHighsNewLowsVolume", "difference", _newhighsnewlowsvolume.Difference);
            Plot("NewHighsNewLowsVolume", "ratio", _newhighsnewlowsvolume.Ratio);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `NewHighsNewLowsVolume` indicator properties:

Indicator History

To get the historical data of the `NewHighsNewLowsVolume` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the

`IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class NewHighsNewLowsVolumeAlgorithm : QCAlgorithm
{
    private Symbol _symbol,_reference;
    private NewHighsNewLowsVolume _nhnlv;

    public override void Initialize()
    {
        _symbol = AddEquity("QQQ", Resolution.Daily).Symbol;
        _reference = AddEquity("SPY", Resolution.Daily).Symbol;
        _nhnlv = NHNLV([_symbol, _reference]);

        var indicatorHistory = IndicatorHistory(_nhnlv, new[] { _symbol, _reference }, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_nhnlv, new[] { _symbol, _reference }, TimeSpan.FromDays(10),
Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_nhnlv, new[] { _symbol, _reference }, new DateTime(2024,
7, 1), new DateTime(2024, 7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var volumeRatio = indicatorHistory.Select(x => ((dynamic)x).VolumeRatio).ToList();
        var difference = indicatorHistory.Select(x => ((dynamic)x).Difference).ToList();
        var ratio = indicatorHistory.Select(x => ((dynamic)x).Ratio).ToList();
    }
}
```

Supported Indicators

Normalized Average True Range

Introduction

This indicator computes the Normalized Average True Range (NATR). The Normalized Average True Range is calculated with the following formula: NATR = (ATR(period) / Close) * 100

To view the implementation of this indicator, see the LEAN GitHub repository .

Using NATR Indicator

To create an automatic indicator for `NormalizedAverageTrueRange` , call the `NATR` helper method from the `QCAlgorithm` class. The `NATR` method creates a `NormalizedAverageTrueRange` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class NormalizedAverageTrueRangeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private NormalizedAverageTrueRange _natr;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _natr = NATR(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_natr.IsReady)
        {
            // The current value of _natr is represented by itself (_natr)
            // or _natr.Current.Value
            Plot("NormalizedAverageTrueRange", "natr", _natr);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `NormalizedAverageTrueRange` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

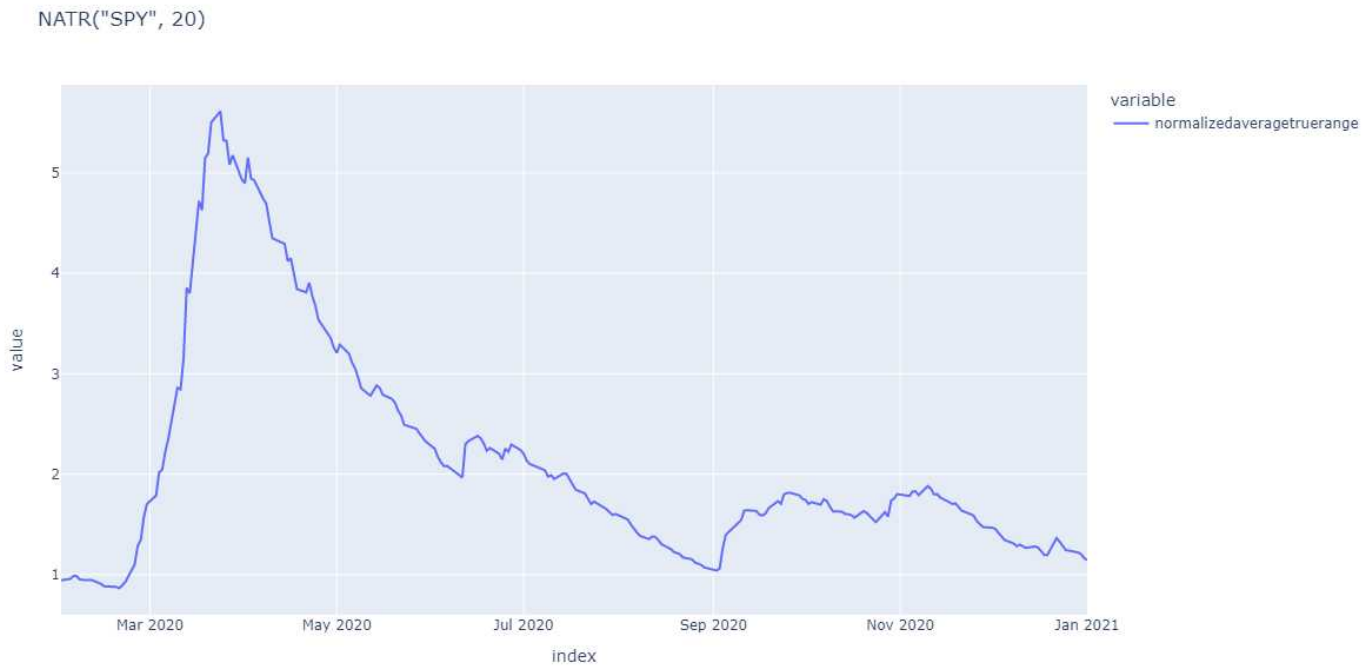
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class NormalizedAverageTrueRangeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private NormalizedAverageTrueRange _normalizedaveragetruerange;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _normalizedaveragetruerange = new NormalizedAverageTrueRange(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _normalizedaveragetruerange.Update(bar);

        if (_normalizedaveragetruerange.IsReady)
        {
            // The current value of _normalizedaveragetruerange is represented by itself
(_normalizedaveragetruerange)
            // or _normalizedaveragetruerange.Current.Value
            Plot("NormalizedAverageTrueRange", "normalizedaveragetruerange", _normalizedaveragetruerange);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `NormalizedAverageTrueRange` indicator properties:



Indicator History

To get the historical data of the `NormalizedAverageTrueRange` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class NormalizedAverageTrueRangeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private NormalizedAverageTrueRange _natr;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _natr = NATR(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_natr, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_natr, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_natr, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

On Balance Volume

Introduction

This indicator computes the On Balance Volume (OBV). The On Balance Volume is calculated by determining the price of the current close price and previous close price. If the current close price is equivalent to the previous price the OBV remains the same, If the current close price is higher the volume of that day is added to the OBV, while a lower close price will result in negative value.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using OBV Indicator

To create an automatic indicator for `OnBalanceVolume` , call the `OBV` helper method from the `QCAlgorithm` class. The `OBV` method creates a `OnBalanceVolume` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class OnBalanceVolumeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private OnBalanceVolume _obv;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _obv = OBV(_symbol);
    }

    public override void OnData(Slice data)
    {

        if (_obv.IsReady)
        {
            // The current value of _obv is represented by itself (_obv)
            // or _obv.Current.Value
            Plot("OnBalanceVolume", "obv", _obv);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `OnBalanceVolume` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

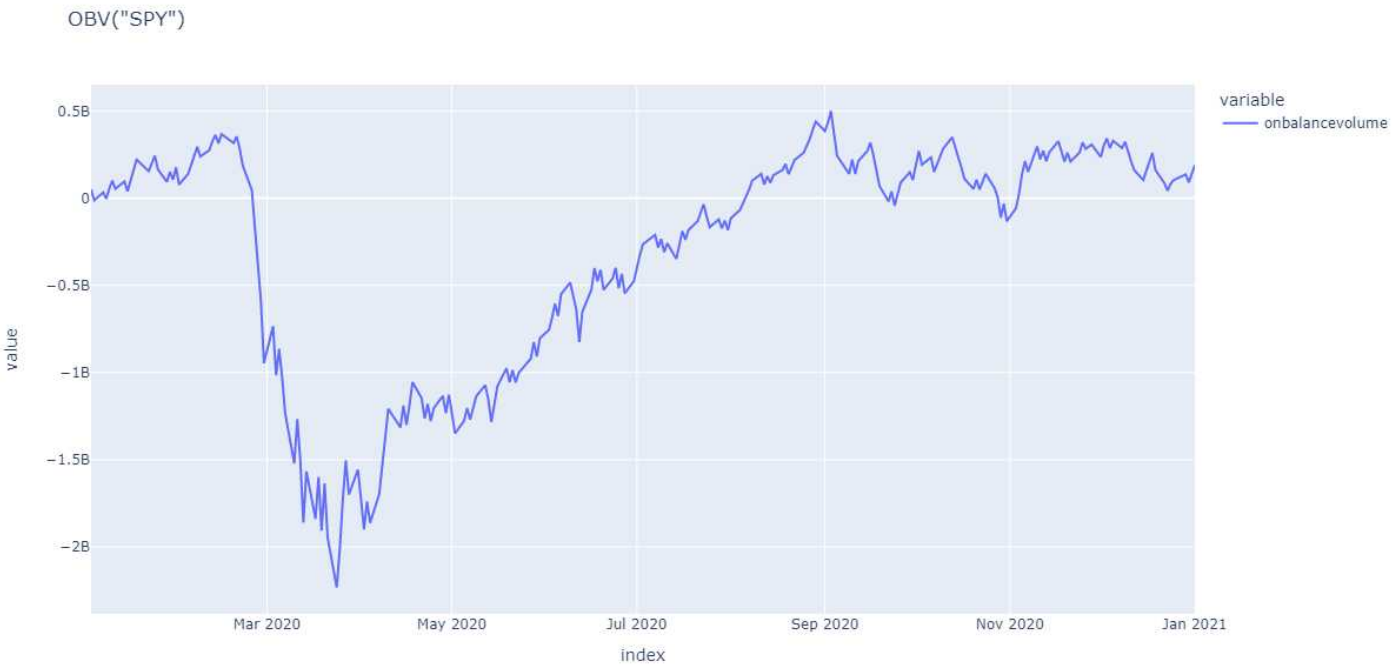
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class OnBalanceVolumeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private OnBalanceVolume _onbalancevolume;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _onbalancevolume = new OnBalanceVolume();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _onbalancevolume.Update(bar);

        if (_onbalancevolume.IsReady)
        {
            // The current value of _onbalancevolume is represented by itself (_onbalancevolume)
            // or _onbalancevolume.Current.Value
            Plot("OnBalanceVolume", "onbalancevolume", _onbalancevolume);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `OnBalanceVolume` indicator properties:



Indicator History

To get the historical data of the `OnBalanceVolume` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class OnBalanceVolumeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private OnBalanceVolume _obv;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _obv = OBV(_symbol);

        var indicatorHistory = IndicatorHistory(_obv, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_obv, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_obv, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Parabolic Stop And Reverse

Introduction

Parabolic SAR Indicator Based on TA-Lib implementation

To view the implementation of this indicator, see the LEAN GitHub repository .

Using PSAR Indicator

To create an automatic indicator for `ParabolicStopAndReverse` , call the `PSAR` helper method from the `QCAlgorithm` class. The `PSAR` method creates a `ParabolicStopAndReverse` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class ParabolicStopAndReverseAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ParabolicStopAndReverse _psar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _psar = PSAR(_symbol, 0.02m, 0.02m, 0.2m);
    }

    public override void OnData(Slice data)
    {

        if (_psar.IsReady)
        {
            // The current value of _psar is represented by itself (_psar)
            // or _psar.Current.Value
            Plot("ParabolicStopAndReverse", "psar", _psar);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `ParabolicStopAndReverse` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

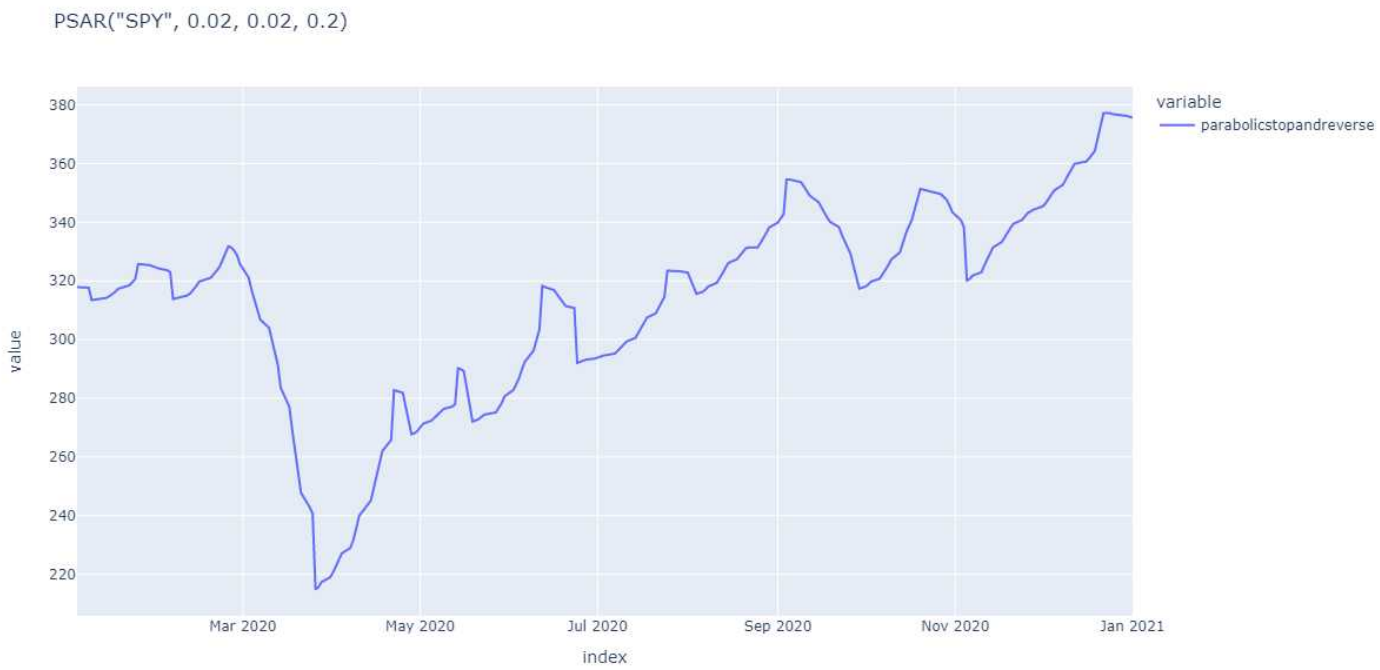
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ParabolicStopAndReverseAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ParabolicStopAndReverse _parabolicstopandreverse;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _parabolicstopandreverse = new ParabolicStopAndReverse(0.02m, 0.02m, 0.2m);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _parabolicstopandreverse.Update(bar);

        if (_parabolicstopandreverse.IsReady)
        {
            // The current value of _parabolicstopandreverse is represented by itself (_parabolicstopandreverse)
            // or _parabolicstopandreverse.Current.Value
            Plot("ParabolicStopAndReverse", "parabolicstopandreverse", _parabolicstopandreverse);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `ParabolicStopAndReverse` indicator properties:



Indicator History

To get the historical data of the `ParabolicStopAndReverse` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class ParabolicStopAndReverseAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ParabolicStopAndReverse _psar;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _psar = PSAR(_symbol, 0.02m, 0.02m, 0.2m);

        var indicatorHistory = IndicatorHistory(_psar, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_psar, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_psar, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Parabolic Stop And Reverse Extended

Introduction

Parabolic SAR Extended Indicator Based on TA-Lib implementation

To view the implementation of this indicator, see the LEAN GitHub repository .

Using SAREXT Indicator

To create an automatic indicator for `ParabolicStopAndReverseExtended` , call the `SAREXT` helper method from the `QCAlgorithm` class. The `SAREXT` method creates a `ParabolicStopAndReverseExtended` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class ParabolicStopAndReverseExtendedAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ParabolicStopAndReverseExtended _sarext;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _sarext = SAREXT(_symbol, 0.0m, 0.0m, 0.02m, 0.02m, 0.2m, 0.02m, 0.02m, 0.2m);
    }

    public override void OnData(Slice data)
    {

        if (_sarext.IsReady)
        {
            // The current value of _sarext is represented by itself (_sarext)
            // or _sarext.Current.Value
            Plot("ParabolicStopAndReverseExtended", "sarext", _sarext);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `ParabolicStopAndReverseExtended` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

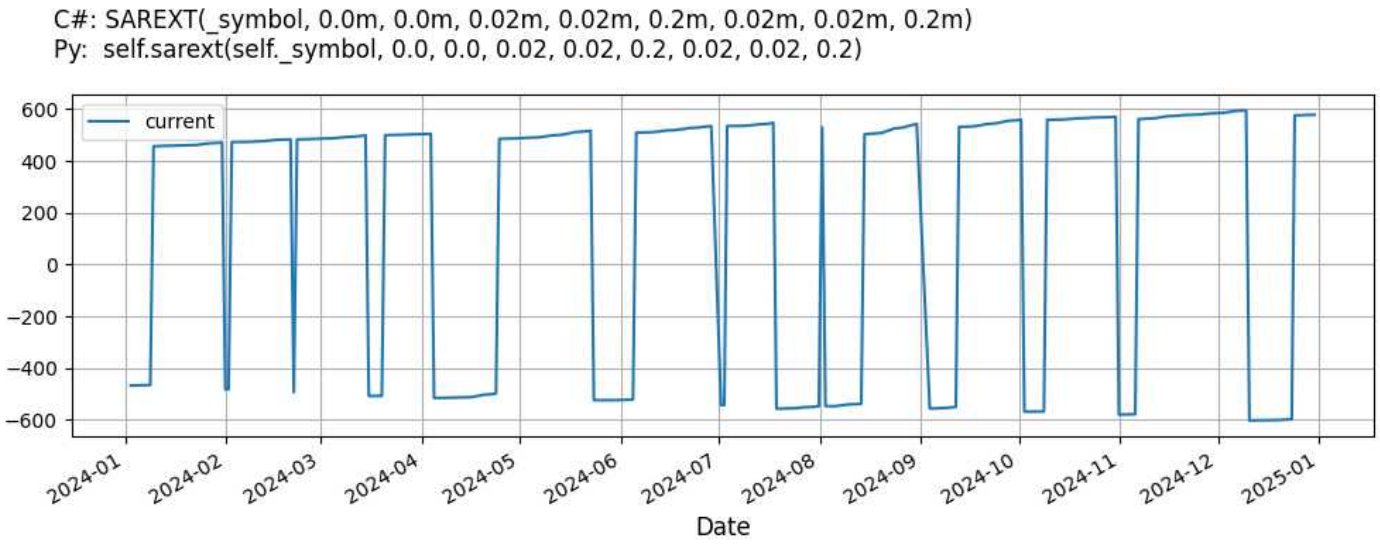
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class ParabolicStopAndReverseExtendedAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ParabolicStopAndReverseExtended _parabolicstopandreverseextended;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _parabolicstopandreverseextended = new ParabolicStopAndReverseExtended(0.0m, 0.0m, 0.02m, 0.02m, 0.2m, 0.02m,
0.02m, 0.2m);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _parabolicstopandreverseextended.Update(bar);

        if (_parabolicstopandreverseextended.IsReady)
        {
            // The current value of _parabolicstopandreverseextended is represented by itself
(_parabolicstopandreverseextended)
            // or _parabolicstopandreverseextended.Current.Value
            Plot("ParabolicStopAndReverseExtended", "parabolicstopandreverseextended",
_parabolicstopandreverseextended);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `ParabolicStopAndReverseExtended` indicator properties:



Indicator History

To get the historical data of the `ParabolicStopAndReverseExtended` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class ParabolicStopAndReverseExtendedAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ParabolicStopAndReverseExtended _sarext;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _sarext = SAREXT(_symbol, 0.0m, 0.0m, 0.02m, 0.02m, 0.2m, 0.02m, 0.02m, 0.2m);

        var indicatorHistory = IndicatorHistory(_sarext, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_sarext, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_sarext, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Percentage Price Oscillator

Introduction

This indicator computes the Percentage Price Oscillator (PPO) The Percentage Price Oscillator is calculated using the following formula: PPO[i] = 100 * (FastMA[i] - SlowMA[i]) / SlowMA[i]

To view the implementation of this indicator, see the LEAN GitHub repository .

Using PPO Indicator

To create an automatic indicator for `PercentagePriceOscillator` , call the `PPO` helper method from the `QCAlgorithm` class. The `PPO` method creates a `PercentagePriceOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

C#

```
public class PercentagePriceOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private PercentagePriceOscillator _ppo;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ppo = PPO(_symbol, 10, 20, MovingAverageType.Simple);
    }

    public override void OnData(Slice data)
    {

        if (_ppo.IsReady)
        {
            // The current value of _ppo is represented by itself (_ppo)
            // or _ppo.Current.Value
            Plot("PercentagePriceOscillator", "ppo", _ppo);
            // Plot all properties of abands
            Plot("PercentagePriceOscillator", "fast", _ppo.Fast);
            Plot("PercentagePriceOscillator", "slow", _ppo.Slow);
            Plot("PercentagePriceOscillator", "signal", _ppo.Signal);
            Plot("PercentagePriceOscillator", "histogram", _ppo.Histogram);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `PercentagePriceOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

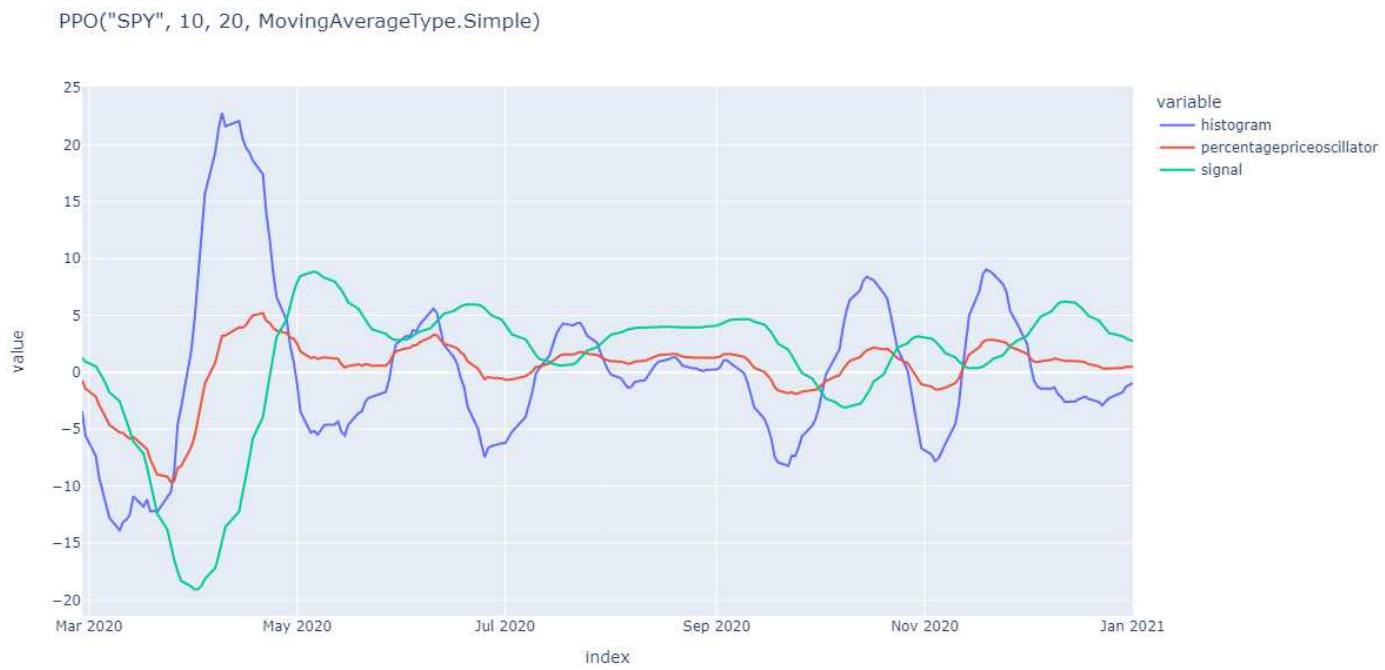
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class PercentagePriceOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private PercentagePriceOscillator _percentagepriceoscillator;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _percentagepriceoscillator = new PercentagePriceOscillator(10, 20, MovingAverageType.Simple);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _percentagepriceoscillator.Update(bar.EndTime, bar.Close);

        if (_percentagepriceoscillator.IsReady)
        {
            // The current value of _percentagepriceoscillator is represented by itself (_percentagepriceoscillator)
            // or _percentagepriceoscillator.Current.Value
            Plot("PercentagePriceOscillator", "percentagepriceoscillator", _percentagepriceoscillator);
            // Plot all properties of abands
            Plot("PercentagePriceOscillator", "fast", _percentagepriceoscillator.Fast);
            Plot("PercentagePriceOscillator", "slow", _percentagepriceoscillator.Slow);
            Plot("PercentagePriceOscillator", "signal", _percentagepriceoscillator.Signal);
            Plot("PercentagePriceOscillator", "histogram", _percentagepriceoscillator.Histogram);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `PercentagePriceOscillator` indicator properties:



Indicator History

To get the historical data of the `PercentagePriceOscillator` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security

subscription.

```C#
public class PercentagePriceOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private PercentagePriceOscillator _ppo;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ppo = PPO(_symbol, 10, 20, MovingAverageType.Simple);

        var indicatorHistory = IndicatorHistory(_ppo, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_ppo, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_ppo, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var fast = indicatorHistory.Select(x => ((dynamic)x).Fast).ToList();
        var slow = indicatorHistory.Select(x => ((dynamic)x).Slow).ToList();
        var signal = indicatorHistory.Select(x => ((dynamic)x).Signal).ToList();
        var histogram = indicatorHistory.Select(x => ((dynamic)x).Histogram).ToList();
    }
}
```

Supported Indicators

Pivot Points High Low

Introduction

Pivot Points (High/Low), also known as Bar Count Reversals, indicator. https://www.fidelity.com/learning-center/trading-investing/technical-analysis/technical-indicator-guide/pivot-points-high-low

To view the implementation of this indicator, see the LEAN GitHub repository .

Using PPHL Indicator

To create an automatic indicator for `PivotPointsHighLow` , call the `PPHL` helper method from the `QCAlgorithm` class. The `PPHL` method creates a `PivotPointsHighLow` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class PivotPointsHighLowAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private PivotPointsHighLow _pphl;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _pphl = PPHL(_symbol, 10, 10, 100);
    }

    public override void OnData(Slice data)
    {

        if (_pphl.IsReady)
        {
            // The current value of _pphl is represented by itself (_pphl)
            // or _pphl.Current.Value
            Plot("PivotPointsHighLow", "pphl", _pphl);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `PivotPointsHighLow` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

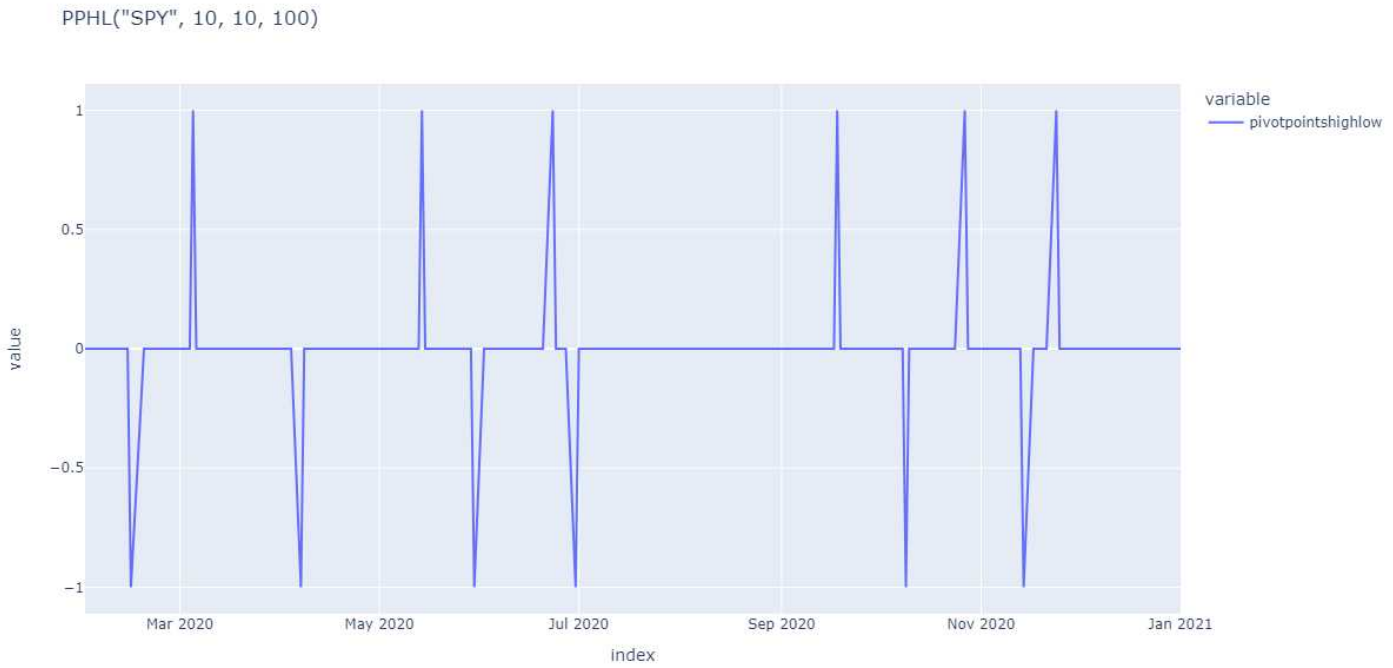
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class PivotPointsHighLowAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private PivotPointsHighLow _pivotpointshighlow;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _pivotpointshighlow = new PivotPointsHighLow(10, 10, 100);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _pivotpointshighlow.Update(bar);

        if (_pivotpointshighlow.IsReady)
        {
            // The current value of _pivotpointshighlow is represented by itself (_pivotpointshighlow)
            // or _pivotpointshighlow.Current.Value
            Plot("PivotPointsHighLow", "pivotpointshighlow", _pivotpointshighlow);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `PivotPointsHighLow` indicator properties:



Indicator History

To get the historical data of the `PivotPointsHighLow` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class PivotPointsHighLowAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private PivotPointsHighLow _pphl;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _pphl = PPHL(_symbol, 10, 10, 100);

        var indicatorHistory = IndicatorHistory(_pphl, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_pphl, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_pphl, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Premier Stochastic Oscillator

Introduction

Premier Stochastic Oscillator (PSO) Indicator implementation. This indicator combines a stochastic oscillator with exponential moving averages to provide a normalized output between -1 and 1, which can be useful for identifying trends and potential reversal points in the market.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using PSO Indicator

To create an automatic indicator for `PremierStochasticOscillator` , call the `PSO` helper method from the `QCAlgorithm` class. The `PSO` method creates a `PremierStochasticOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class PremierStochasticOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private PremierStochasticOscillator _pso;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _pso = PSO(_symbol, 14, 3);
    }

    public override void OnData(Slice data)
    {

        if (_pso.IsReady)
        {
            // The current value of _pso is represented by itself (_pso)
            // or _pso.Current.Value
            Plot("PremierStochasticOscillator", "pso", _pso);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `PremierStochasticOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

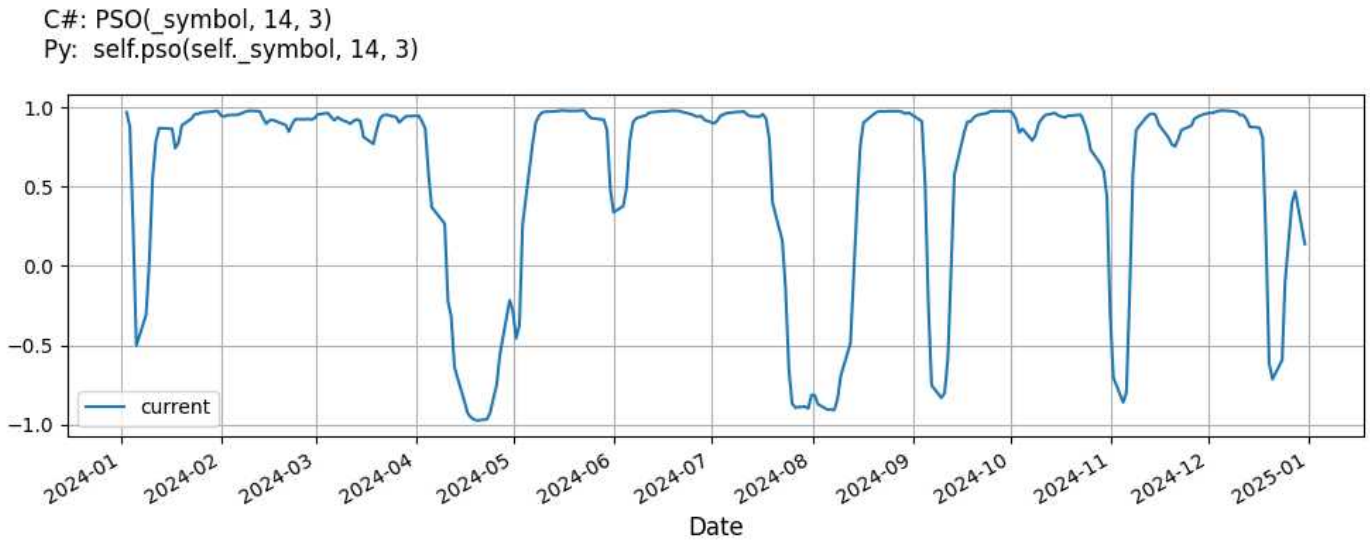
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class PremierStochasticOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private PremierStochasticOscillator _premierstochasticoscillator;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _premierstochasticoscillator = new PremierStochasticOscillator(14, 3);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _premierstochasticoscillator.Update(bar);

        if (_premierstochasticoscillator.IsReady)
        {
            // The current value of _premierstochasticoscillator is represented by itself
(_premierstochasticoscillator)
            // or _premierstochasticoscillator.Current.Value
            Plot("PremierStochasticOscillator", "premierstochasticoscillator", _premierstochasticoscillator);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `PremierStochasticOscillator` indicator properties:



Indicator History

To get the historical data of the `PremierStochasticOscillator` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class PremierStochasticOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private PremierStochasticOscillator _pso;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _pso = PSO(_symbol, 14, 3);

        var indicatorHistory = IndicatorHistory(_pso, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_pso, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_pso, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Rate Of Change

Introduction

This indicator computes the n-period rate of change in a value using the following: (value_0 - value_n) / value_n

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ROC Indicator

To create an automatic indicator for `RateOfChange` , call the `ROC` helper method from the `QCAlgorithm` class. The `ROC` method creates a `RateOfChange` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class RateOfChangeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RateOfChange _roc;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _roc = ROC(_symbol, 10);
    }

    public override void OnData(Slice data)
    {

        if (_roc.IsReady)
        {
            // The current value of _roc is represented by itself (_roc)
            // or _roc.Current.Value
            Plot("RateOfChange", "roc", _roc);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `RateOfChange` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class RateOfChangeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RateOfChange _rateofchange;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rateofchange = new RateOfChange(10);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _rateofchange.Update(bar.EndTime, bar.Close);

        if (_rateofchange.IsReady)
        {
            // The current value of _rateofchange is represented by itself (_rateofchange)
            // or _rateofchange.Current.Value
            Plot("RateOfChange", "rateofchange", _rateofchange);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `RateOfChange` indicator properties:

Indicator History

To get the historical data of the `RateOfChange` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class RateOfChangeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RateOfChange _roc;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _roc = ROC(_symbol, 10);

        var indicatorHistory = IndicatorHistory(_roc, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_roc, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_roc, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Rate Of Change Percent

Introduction

This indicator computes the n-period percentage rate of change in a value using the following: 100 * (value_0 - value_n) / value_n

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ROCP Indicator

To create an automatic indicator for `RateOfChangePercent` , call the `ROCP` helper method from the `QCAlgorithm` class. The `ROCP` method creates a `RateOfChangePercent` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class RateOfChangePercentAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RateOfChangePercent _rocp;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rocp = ROCP(_symbol, 10);
    }

    public override void OnData(Slice data)
    {

        if (_rocp.IsReady)
        {
            // The current value of _rocp is represented by itself (_rocp)
            // or _rocp.Current.Value
            Plot("RateOfChangePercent", "rocp", _rocp);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `RateOfChangePercent` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class RateOfChangePercentAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RateOfChangePercent _rateofchangepercent;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rateofchangepercent = new RateOfChangePercent(10);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _rateofchangepercent.Update(bar.EndTime, bar.Close);

        if (_rateofchangepercent.IsReady)
        {
            // The current value of _rateofchangepercent is represented by itself (_rateofchangepercent)
            // or _rateofchangepercent.Current.Value
            Plot("RateOfChangePercent", "rateofchangepercent", _rateofchangepercent);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `RateOfChangePercent` indicator properties:

Indicator History

To get the historical data of the `RateOfChangePercent` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class RateOfChangePercentAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RateOfChangePercent _rocp;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rocp = ROCP(_symbol, 10);

        var indicatorHistory = IndicatorHistory(_rocp, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_rocp, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_rocp, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Rate Of Change Ratio

Introduction

This indicator computes the Rate Of Change Ratio (ROCR). The Rate Of Change Ratio is calculated with the following formula: ROCR = price / prevPrice

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ROCR Indicator

To create an automatic indicator for `RateOfChangeRatio` , call the `ROCR` helper method from the `QCAlgorithm` class. The `ROCR` method creates a `RateOfChangeRatio` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class RateOfChangeRatioAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RateOfChangeRatio _rocr;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rocr = ROCR(_symbol, 10);
    }

    public override void OnData(Slice data)
    {

        if (_rocr.IsReady)
        {
            // The current value of _rocr is represented by itself (_rocr)
            // or _rocr.Current.Value
            Plot("RateOfChangeRatio", "rocr", _rocr);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `RateOfChangeRatio` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

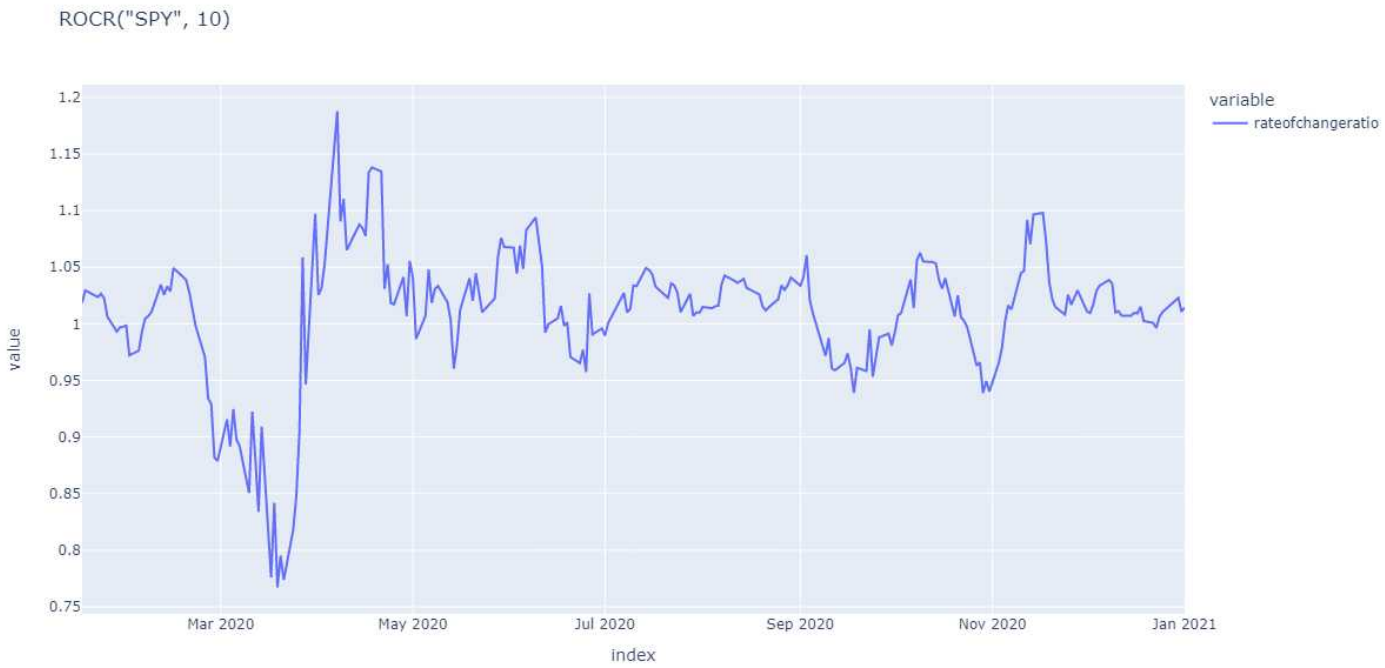
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class RateOfChangeRatioAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RateOfChangeRatio _rateofchangeratio;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rateofchangeratio = new RateOfChangeRatio(10);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _rateofchangeratio.Update(bar.EndTime, bar.Close);

        if (_rateofchangeratio.IsReady)
        {
            // The current value of _rateofchangeratio is represented by itself (_rateofchangeratio)
            // or _rateofchangeratio.Current.Value
            Plot("RateOfChangeRatio", "rateofchangeratio", _rateofchangeratio);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `RateOfChangeRatio` indicator properties:



Indicator History

To get the historical data of the `RateOfChangeRatio` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class RateOfChangeRatioAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RateOfChangeRatio _rocr;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rocr = ROCR(_symbol, 10);

        var indicatorHistory = IndicatorHistory(_rocr, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_rocr, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_rocr, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Regression Channel

Introduction

The Regression Channel indicator extends the with the inclusion of two (upper and lower) channel lines that are distanced from the linear regression line by a user defined number of standard deviations. Reference: http://www.onlinetradingconcepts.com/TechnicalAnalysis/LinRegChannel.html

To view the implementation of this indicator, see the LEAN GitHub repository .

Using RC Indicator

To create an automatic indicator for `RegressionChannel` , call the `RC` helper method from the `QCAlgorithm` class. The `RC` method creates a `RegressionChannel` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class RegressionChannelAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RegressionChannel _rc;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rc = RC(_symbol, 20, 2);
    }

    public override void OnData(Slice data)
    {

        if (_rc.IsReady)
        {
            // The current value of _rc is represented by itself (_rc)
            // or _rc.Current.Value
            Plot("RegressionChannel", "rc", _rc);
            // Plot all properties of abands
            Plot("RegressionChannel", "linearregression", _rc.LinearRegression);
            Plot("RegressionChannel", "upperchannel", _rc.UpperChannel);
            Plot("RegressionChannel", "lowerchannel", _rc.LowerChannel);
            Plot("RegressionChannel", "intercept", _rc.Intercept);
            Plot("RegressionChannel", "slope", _rc.Slope);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `RegressionChannel` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class RegressionChannelAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RegressionChannel _regressionchannel;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _regressionchannel = new RegressionChannel(20, 2);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _regressionchannel.Update(bar.EndTime, bar.Close);

        if (_regressionchannel.IsReady)
        {
            // The current value of _regressionchannel is represented by itself (_regressionchannel)
            // or _regressionchannel.Current.Value
            Plot("RegressionChannel", "regressionchannel", _regressionchannel);
            // Plot all properties of abands
            Plot("RegressionChannel", "linearregression", _regressionchannel.LinearRegression);
            Plot("RegressionChannel", "upperchannel", _regressionchannel.UpperChannel);
            Plot("RegressionChannel", "lowerchannel", _regressionchannel.LowerChannel);
            Plot("RegressionChannel", "intercept", _regressionchannel.Intercept);
            Plot("RegressionChannel", "slope", _regressionchannel.Slope);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `RegressionChannel` indicator properties:

Indicator History

To get the historical data of the `RegressionChannel` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period

of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class RegressionChannelAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RegressionChannel _rc;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rc = RC(_symbol, 20, 2);

        var indicatorHistory = IndicatorHistory(_rc, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_rc, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_rc, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var linearRegression = indicatorHistory.Select(x => ((dynamic)x).LinearRegression).ToList();
        var upperChannel = indicatorHistory.Select(x => ((dynamic)x).UpperChannel).ToList();
        var lowerChannel = indicatorHistory.Select(x => ((dynamic)x).LowerChannel).ToList();
        var intercept = indicatorHistory.Select(x => ((dynamic)x).Intercept).ToList();
        var slope = indicatorHistory.Select(x => ((dynamic)x).Slope).ToList();
    }
}
```

Supported Indicators

Relative Daily Volume

Introduction

The Relative Daily Volume indicator is an indicator that compares current cumulative volume to the cumulative volume for a given time of day, measured as a ratio. Current volume from open to current time of day / Average over the past x days from open to current time of day

To view the implementation of this indicator, see the LEAN GitHub repository .

Using RDV Indicator

To create an automatic indicator for `RelativeDailyVolume` , call the `RDV` helper method from the `QCAlgorithm` class. The `RDV` method creates a `RelativeDailyVolume` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

C#

```
public class RelativeDailyVolumeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RelativeDailyVolume _rdv;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rdv = RDV(_symbol, 2);
    }

    public override void OnData(Slice data)
    {

        if (_rdv.IsReady)
        {
            // The current value of _rdv is represented by itself (_rdv)
            // or _rdv.Current.Value
            Plot("RelativeDailyVolume", "rdv", _rdv);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `RelativeDailyVolume` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

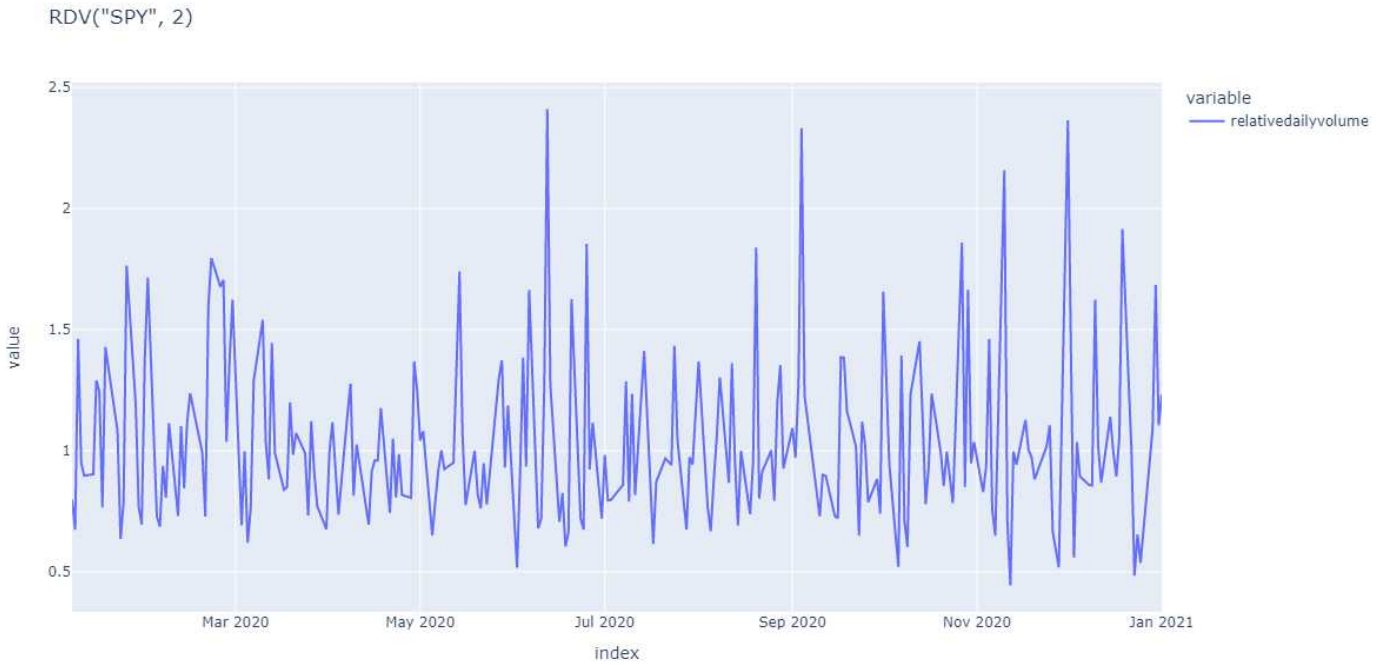
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class RelativeDailyVolumeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RelativeDailyVolume _relativedailyvolume;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _relativedailyvolume = new RelativeDailyVolume(2);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _relativedailyvolume.Update(bar);

        if (_relativedailyvolume.IsReady)
        {
            // The current value of _relativedailyvolume is represented by itself (_relativedailyvolume)
            // or _relativedailyvolume.Current.Value
            Plot("RelativeDailyVolume", "relativedailyvolume", _relativedailyvolume);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `RelativeDailyVolume` indicator properties:



Indicator History

To get the historical data of the `RelativeDailyVolume` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class RelativeDailyVolumeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RelativeDailyVolume _rdv;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rdv = RDV(_symbol, 2);

        var indicatorHistory = IndicatorHistory(_rdv, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_rdv, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_rdv, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Relative Moving Average

Introduction

This indicator represents the relative moving average indicator (RMA). RMA = SMA(3 x Period) - SMA(2 x Period) + SMA(1 x Period) per formula: https://www.hybrid-solutions.com/plugins/client-vtl-plugins/free/rma.html

To view the implementation of this indicator, see the LEAN GitHub repository .

Using RMA Indicator

To create an automatic indicator for `RelativeMovingAverage` , call the `RMA` helper method from the `QCAlgorithm` class. The `RMA` method creates a `RelativeMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class RelativeMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RelativeMovingAverage _rma;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rma = RMA(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_rma.IsReady)
        {
            // The current value of _rma is represented by itself (_rma)
            // or _rma.Current.Value
            Plot("RelativeMovingAverage", "rma", _rma);
            // Plot all properties of abands
            Plot("RelativeMovingAverage", "shortaverage", _rma.ShortAverage);
            Plot("RelativeMovingAverage", "mediumaverage", _rma.MediumAverage);
            Plot("RelativeMovingAverage", "longaverage", _rma.LongAverage);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `RelativeMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

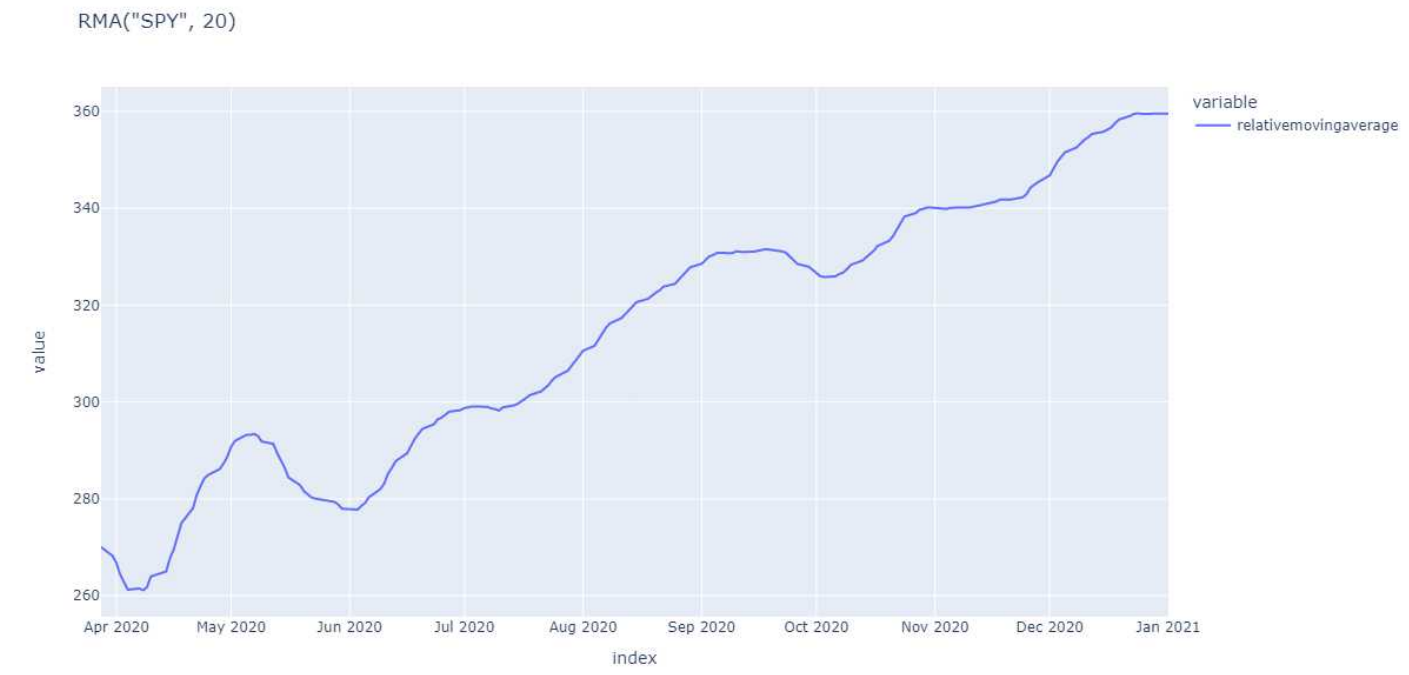
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class RelativeMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RelativeMovingAverage _relativemovingaverage;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _relativemovingaverage = new RelativeMovingAverage(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _relativemovingaverage.Update(bar.EndTime, bar.Close);

        if (_relativemovingaverage.IsReady)
        {
            // The current value of _relativemovingaverage is represented by itself (_relativemovingaverage)
            // or _relativemovingaverage.Current.Value
            Plot("RelativeMovingAverage", "relativemovingaverage", _relativemovingaverage);
            // Plot all properties of abands
            Plot("RelativeMovingAverage", "shortaverage", _relativemovingaverage.ShortAverage);
            Plot("RelativeMovingAverage", "mediumaverage", _relativemovingaverage.MediumAverage);
            Plot("RelativeMovingAverage", "longaverage", _relativemovingaverage.LongAverage);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `RelativeMovingAverage` indicator properties:



Indicator History

To get the historical data of the `RelativeMovingAverage` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class RelativeMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RelativeMovingAverage _rma;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rma = RMA(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_rma, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_rma, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_rma, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var shortAverage = indicatorHistory.Select(x => ((dynamic)x).ShortAverage).ToList();
        var mediumAverage = indicatorHistory.Select(x => ((dynamic)x).MediumAverage).ToList();
        var longAverage = indicatorHistory.Select(x => ((dynamic)x).LongAverage).ToList();
    }
}
```

Supported Indicators

Relative Strength Index

Introduction

This indicator represents the Relative Strength Index (RSI) developed by K. Welles Wilder. You can optionally specified a different moving average type to be used in the computation

To view the implementation of this indicator, see the LEAN GitHub repository .

Using RSI Indicator

To create an automatic indicator for `RelativeStrengthIndex` , call the `RSI` helper method from the `QCAlgorithm` class. The `RSI` method creates a `RelativeStrengthIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class RelativeStrengthIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RelativeStrengthIndex _rsi;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rsi = RSI(_symbol, 14);
    }

    public override void OnData(Slice data)
    {

        if (_rsi.IsReady)
        {
            // The current value of _rsi is represented by itself (_rsi)
            // or _rsi.Current.Value
            Plot("RelativeStrengthIndex", "rsi", _rsi);
            // Plot all properties of abands
            Plot("RelativeStrengthIndex", "averageloss", _rsi.AverageLoss);
            Plot("RelativeStrengthIndex", "averagegain", _rsi.AverageGain);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `RelativeStrengthIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

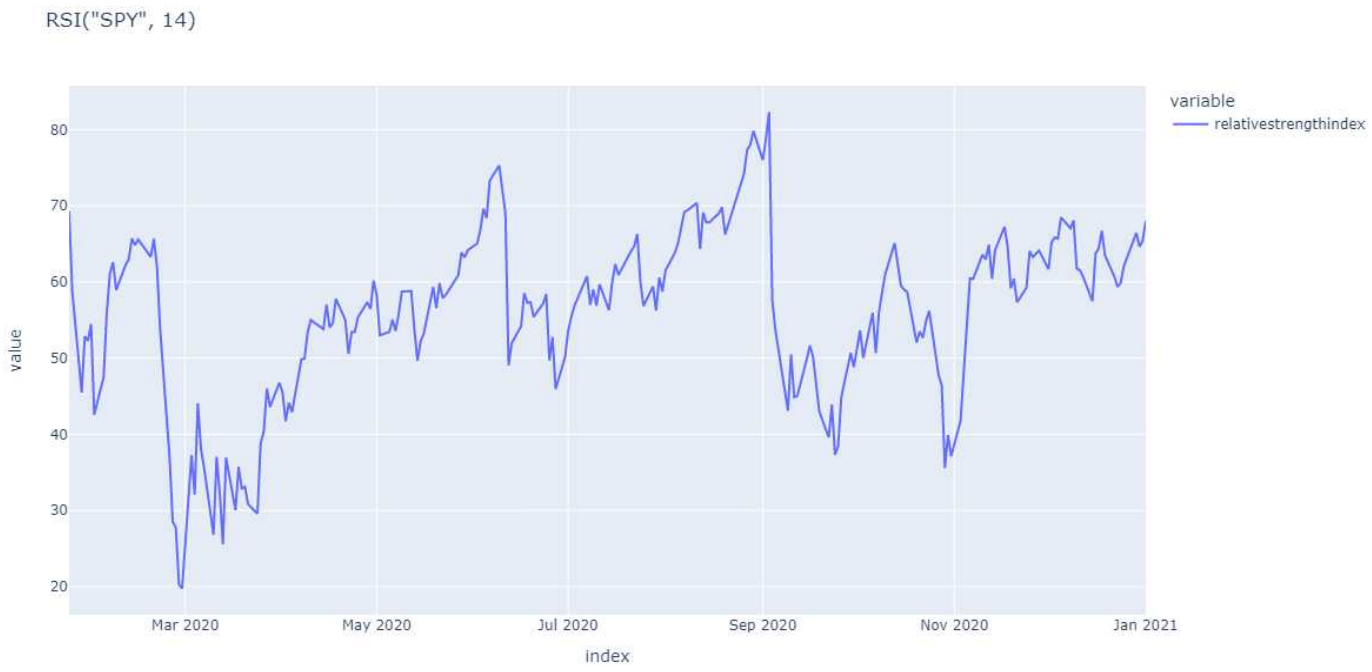
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class RelativeStrengthIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RelativeStrengthIndex _relativestrengthindex;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _relativestrengthindex = new RelativeStrengthIndex(14);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _relativestrengthindex.Update(bar.EndTime, bar.Close);

        if (_relativestrengthindex.IsReady)
        {
            // The current value of _relativestrengthindex is represented by itself (_relativestrengthindex)
            // or _relativestrengthindex.Current.Value
            Plot("RelativeStrengthIndex", "relativestrengthindex", _relativestrengthindex);
            // Plot all properties of abands
            Plot("RelativeStrengthIndex", "averageloss", _relativestrengthindex.AverageLoss);
            Plot("RelativeStrengthIndex", "averagegain", _relativestrengthindex.AverageGain);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `RelativeStrengthIndex` indicator properties:



Indicator History

To get the historical data of the `RelativeStrengthIndex` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class RelativeStrengthIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RelativeStrengthIndex _rsi;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rsi = RSI(_symbol, 14);

        var indicatorHistory = IndicatorHistory(_rsi, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_rsi, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_rsi, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var averageLoss = indicatorHistory.Select(x => ((dynamic)x).AverageLoss).ToList();
        var averageGain = indicatorHistory.Select(x => ((dynamic)x).AverageGain).ToList();
    }
}
```

Supported Indicators

Relative Vigor Index

Introduction

The Relative Vigor Index (RVI) compares the ratio of the closing price of a security to its trading range. For illustration, let: a = Close−Open b = Close−Open of One Bar Prior to a c = Close−Open of One Bar Prior to b d = Close−Open of One Bar Prior to c e = High−Low of Bar a f = High−Low of Bar b g = High−Low of Bar c h = High−Low of Bar d Then let (a+2*(b+c)+d)/6 be NUM and (e+2*(f+g)+h)/6 be DENOM. RVI = SMA(NUM)/SMA(DENOM) for a specified period.
https://www.investopedia.com/terms/r/relative_vigor_index.asp

To view the implementation of this indicator, see the LEAN GitHub repository .

Using RVI Indicator

To create an automatic indicator for `RelativeVigorIndex` , call the `RVI` helper method from the `QCAlgorithm` class. The `RVI` method creates a `RelativeVigorIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class RelativeVigorIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RelativeVigorIndex _rvi;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rvi = RVI(_symbol, 20, MovingAverageType.Simple);
    }

    public override void OnData(Slice data)
    {

        if (_rvi.IsReady)
        {
            // The current value of _rvi is represented by itself (_rvi)
            // or _rvi.Current.Value
            Plot("RelativeVigorIndex", "rvi", _rvi);
            // Plot all properties of abands
            Plot("RelativeVigorIndex", "signal", _rvi.Signal);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `RelativeVigorIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

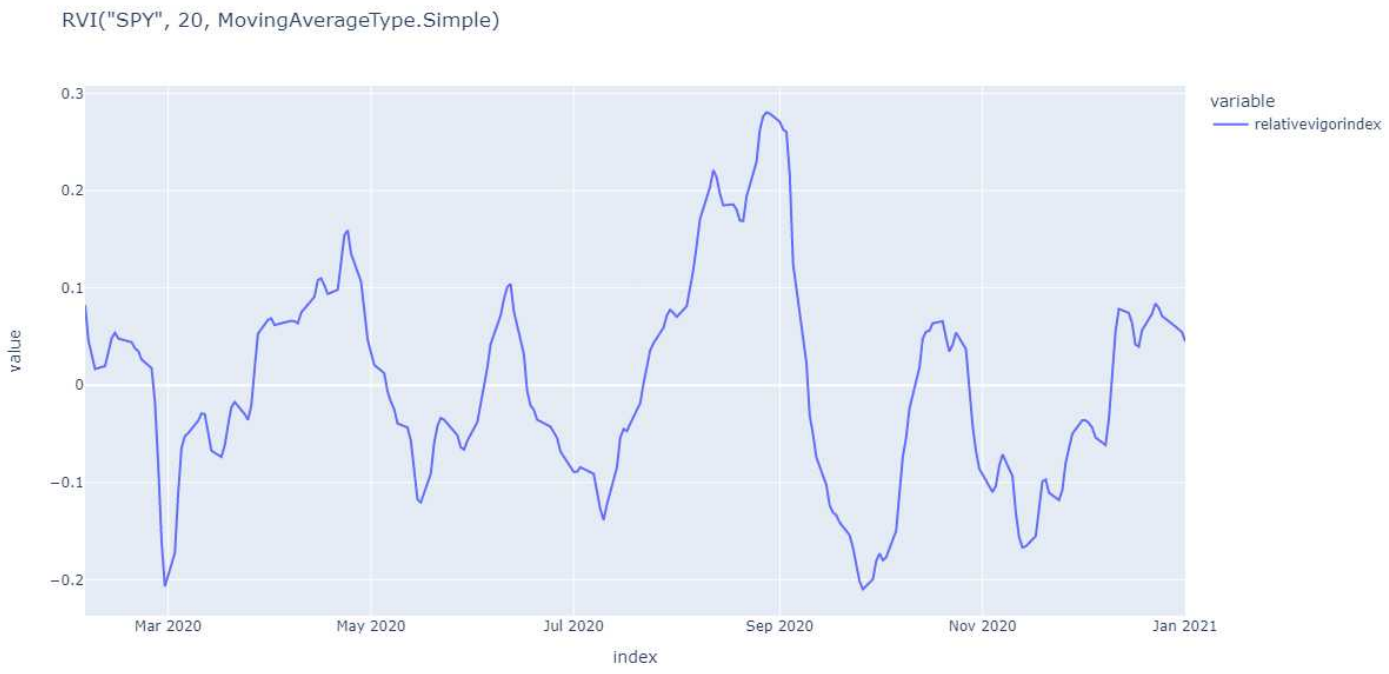
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class RelativeVigorIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RelativeVigorIndex _relativevigorindex;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _relativevigorindex = new RelativeVigorIndex(20, MovingAverageType.Simple);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _relativevigorindex.Update(bar);

        if (_relativevigorindex.IsReady)
        {
            // The current value of _relativevigorindex is represented by itself (_relativevigorindex)
            // or _relativevigorindex.Current.Value
            Plot("RelativeVigorIndex", "relativevigorindex", _relativevigorindex);
            // Plot all properties of abands
            Plot("RelativeVigorIndex", "signal", _relativevigorindex.Signal);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `RelativeVigorIndex` indicator properties:



Indicator History

To get the historical data of the `RelativeVigorIndex` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class RelativeVigorIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RelativeVigorIndex _rvi;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rvi = RVI(_symbol, 20, MovingAverageType.Simple);

        var indicatorHistory = IndicatorHistory(_rvi, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_rvi, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_rvi, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var signal = indicatorHistory.Select(x => ((dynamic)x).Signal).ToList();
    }
}
```

Supported Indicators

Rho

Introduction

Option Rho indicator that calculate the rho of an option

To view the implementation of this indicator, see the LEAN GitHub repository .

Using R Indicator

To create an automatic indicator for `Rho` , call the `R` helper method from the `QCAlgorithm` class. The `R` method creates a `Rho` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class RhoAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _option, _mirrorOption;
    private Rho _r;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 20);

        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _option = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920C00564000");
        _mirrorOption = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920P00564000");
        AddOptionContract(_option, Resolution.Daily);
        AddOptionContract(_mirrorOption, Resolution.Daily);

        _r = R(_option, _mirrorOption);
    }

    public override void OnData(Slice data)
    {
        if (_r.IsReady)
        {
            // The current value of _r is represented by itself (_r)
            // or _r.Current.Value
            Plot("Rho", "r", _r);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `Rho` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

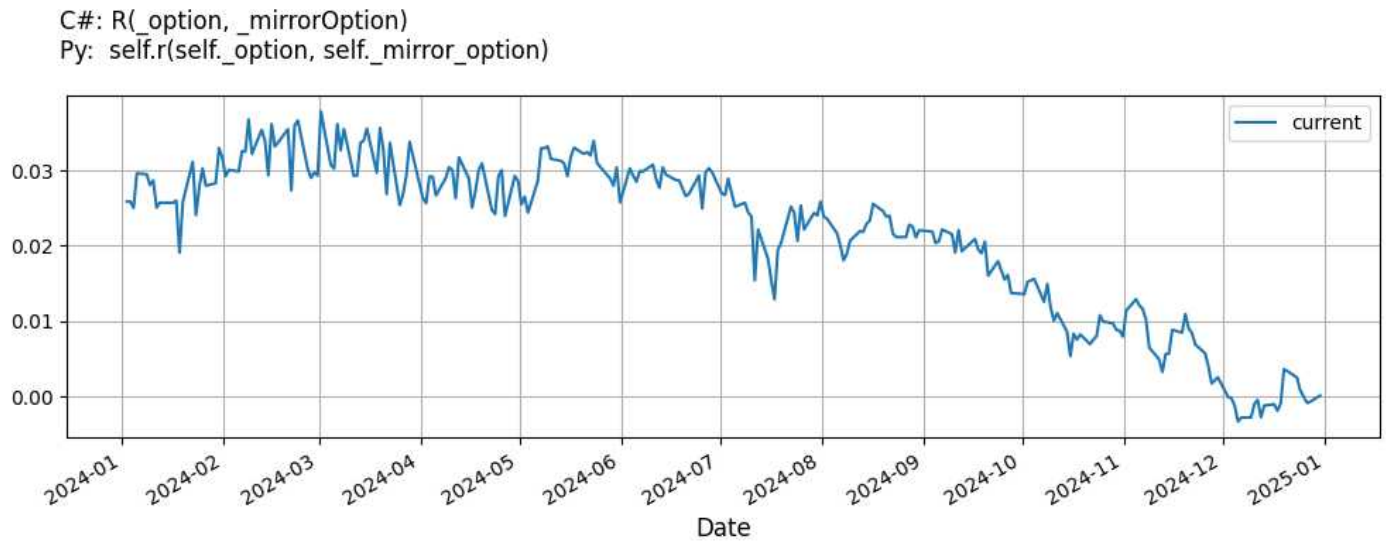
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class RhoAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _option, _mirrorOption;
    private Rho _rho;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 20);

        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        var interestRateModel = new InterestRateProvider();
        var dividendYieldModel = new DividendYieldProvider(_symbol);

        _option = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920C00564000");
        _mirrorOption = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920P00564000");
        AddOptionContract(_option, Resolution.Daily);
        AddOptionContract(_mirrorOption, Resolution.Daily);

        _rho = new Rho(_option, interestRateModel, dividendYieldModel, _mirrorOption);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _rho.Update(new IndicatorDataPoint(_symbol, bar.EndTime, bar.Close));
        if (data.QuoteBars.TryGetValue(_option, out var quoteBar))
            _rho.Update(new IndicatorDataPoint(_option, quoteBar.EndTime, quoteBar.Close));
        if (data.QuoteBars.TryGetValue(_mirrorOption, out quoteBar))
            _rho.Update(new IndicatorDataPoint(_mirrorOption, quoteBar.EndTime, quoteBar.Close));

        if (_rho.IsReady)
        {
            // The current value of _rho is represented by itself (_rho)
            // or _rho.Current.Value
            Plot("Rho", "rho", _rho);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `Rho` indicator properties:



Indicator History

To get the historical data of the `Rho` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory`

method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class RhoAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _option, _mirrorOption;
    private Rho _r;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 20);

        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _option = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920C00564000");
        _mirrorOption = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920P00564000");
        AddOptionContract(_option, Resolution.Daily);
        AddOptionContract(_mirrorOption, Resolution.Daily);

        _r = R(_option, _mirrorOption);

        var indicatorHistory = IndicatorHistory(_r, new[] { _symbol, _option, _mirrorOption }, 100,
Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_r, new[] { _symbol, _option, _mirrorOption },
TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_r, new[] { _symbol, _option, _mirrorOption }, new
DateTime(2024, 7, 1), new DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Rogers Satchell Volatility

Introduction

This indicator computes the Rogers-Satchell Volatility It is an estimator for measuring the volatility of securities with an average return not equal to zero.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using RSV Indicator

To create an automatic indicator for `RogersSatchellVolatility` , call the `RSV` helper method from the `QCAlgorithm` class. The `RSV` method creates a `RogersSatchellVolatility` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class RogersSatchellVolatilityAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RogersSatchellVolatility _rsv;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rsv = RSV(_symbol, 30);
    }

    public override void OnData(Slice data)
    {

        if (_rsv.IsReady)
        {
            // The current value of _rsv is represented by itself (_rsv)
            // or _rsv.Current.Value
            Plot("RogersSatchellVolatility", "rsv", _rsv);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `RogersSatchellVolatility` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

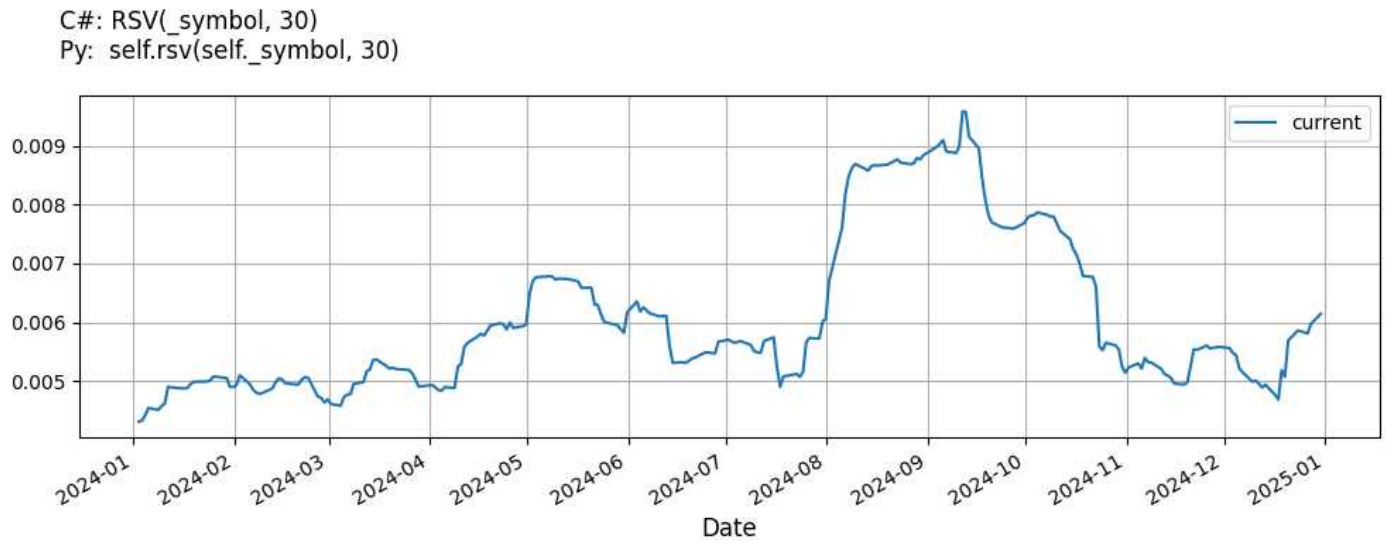
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class RogersSatchellVolatilityAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RogersSatchellVolatility _rogerssatchellvolatility;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rogerssatchellvolatility = new RogersSatchellVolatility(30);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _rogerssatchellvolatility.Update(bar);

        if (_rogerssatchellvolatility.IsReady)
        {
            // The current value of _rogerssatchellvolatility is represented by itself (_rogerssatchellvolatility)
            // or _rogerssatchellvolatility.Current.Value
            Plot("RogersSatchellVolatility", "rogerssatchellvolatility", _rogerssatchellvolatility);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `RogersSatchellVolatility` indicator properties:



Indicator History

To get the historical data of the `RogersSatchellVolatility` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class RogersSatchellVolatilityAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private RogersSatchellVolatility _rsv;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _rsv = RSV(_symbol, 30);

        var indicatorHistory = IndicatorHistory(_rsv, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_rsv, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_rsv, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Schaff Trend Cycle

Introduction

This indicator creates the Schaff Trend Cycle

To view the implementation of this indicator, see the LEAN GitHub repository .

Using STC Indicator

To create an automatic indicator for `SchaffTrendCycle` , call the `STC` helper method from the `QCAlgorithm` class. The `STC` method creates a `SchaffTrendCycle` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class SchaffTrendCycleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SchaffTrendCycle _stc;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _stc = STC(_symbol, 5, 10, 20, MovingAverageType.Exponential);
    }

    public override void OnData(Slice data)
    {

        if (_stc.IsReady)
        {
            // The current value of _stc is represented by itself (_stc)
            // or _stc.Current.Value
            Plot("SchaffTrendCycle", "stc", _stc);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `SchaffTrendCycle` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

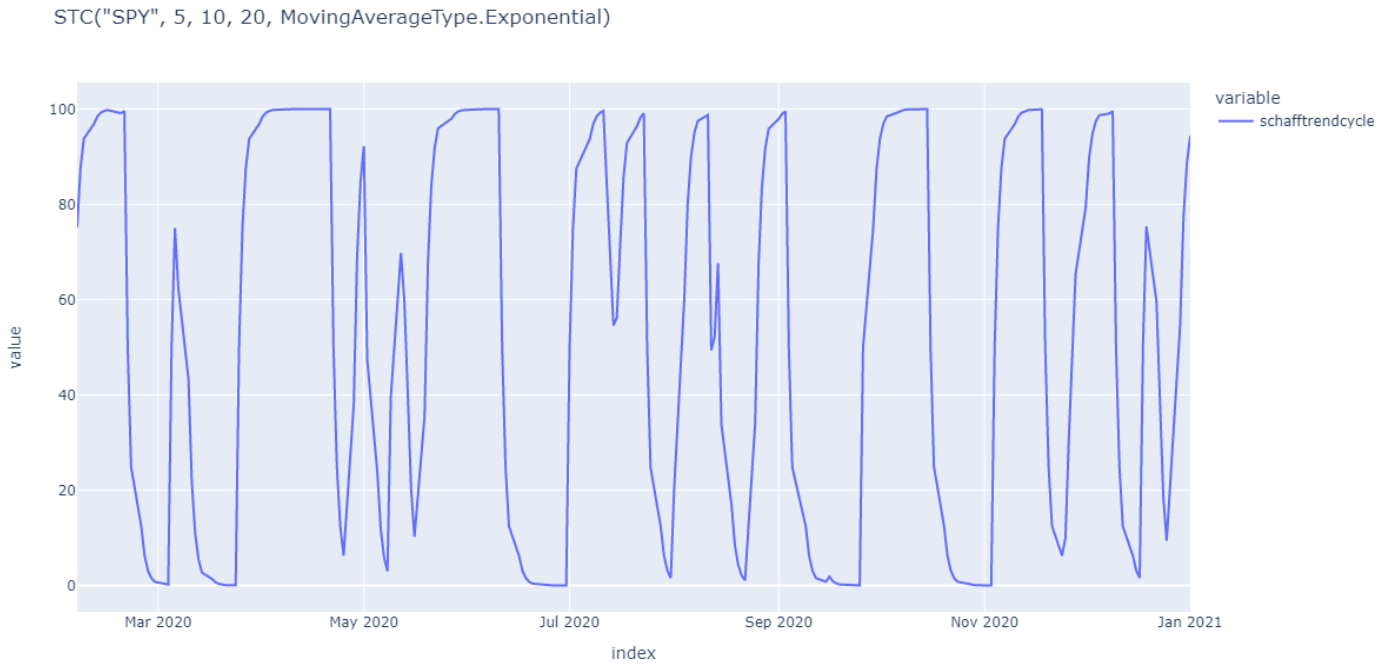
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class SchaffTrendCycleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SchaffTrendCycle _schafftrendcycle;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _schafftrendcycle = new SchaffTrendCycle(5, 10, 20, MovingAverageType.Exponential);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _schafftrendcycle.Update(bar.EndTime, bar.Close);

        if (_schafftrendcycle.IsReady)
        {
            // The current value of _schafftrendcycle is represented by itself (_schafftrendcycle)
            // or _schafftrendcycle.Current.Value
            Plot("SchaffTrendCycle", "schafftrendcycle", _schafftrendcycle);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `SchaffTrendCycle` indicator properties:



Indicator History

To get the historical data of the `SchaffTrendCycle` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class SchaffTrendCycleAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SchaffTrendCycle _stc;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _stc = STC(_symbol, 5, 10, 20, MovingAverageType.Exponential);

        var indicatorHistory = IndicatorHistory(_stc, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_stc, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_stc, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Sharpe Ratio

Introduction

Calculation of the Sharpe Ratio (SR) developed by William F. Sharpe. Reference: https://www.investopedia.com/articles/07/sharpe_ratio.asp Formula: S(x) = (Rx - Rf) / stdDev(Rx) Where: S(x) - sharpe ratio of x Rx - average rate of return for x Rf - risk-free rate

To view the implementation of this indicator, see the LEAN GitHub repository .

Using SR Indicator

To create an automatic indicator for `SharpeRatio` , call the `SR` helper method from the `QCAlgorithm` class. The `SR` method creates a `SharpeRatio` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class SharpeRatioAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SharpeRatio _sr;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _sr = SR(_symbol, 22, 0.03m);
    }

    public override void OnData(Slice data)
    {

        if (_sr.IsReady)
        {
            // The current value of _sr is represented by itself (_sr)
            // or _sr.Current.Value
            Plot("SharpeRatio", "sr", _sr);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `SharpeRatio` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

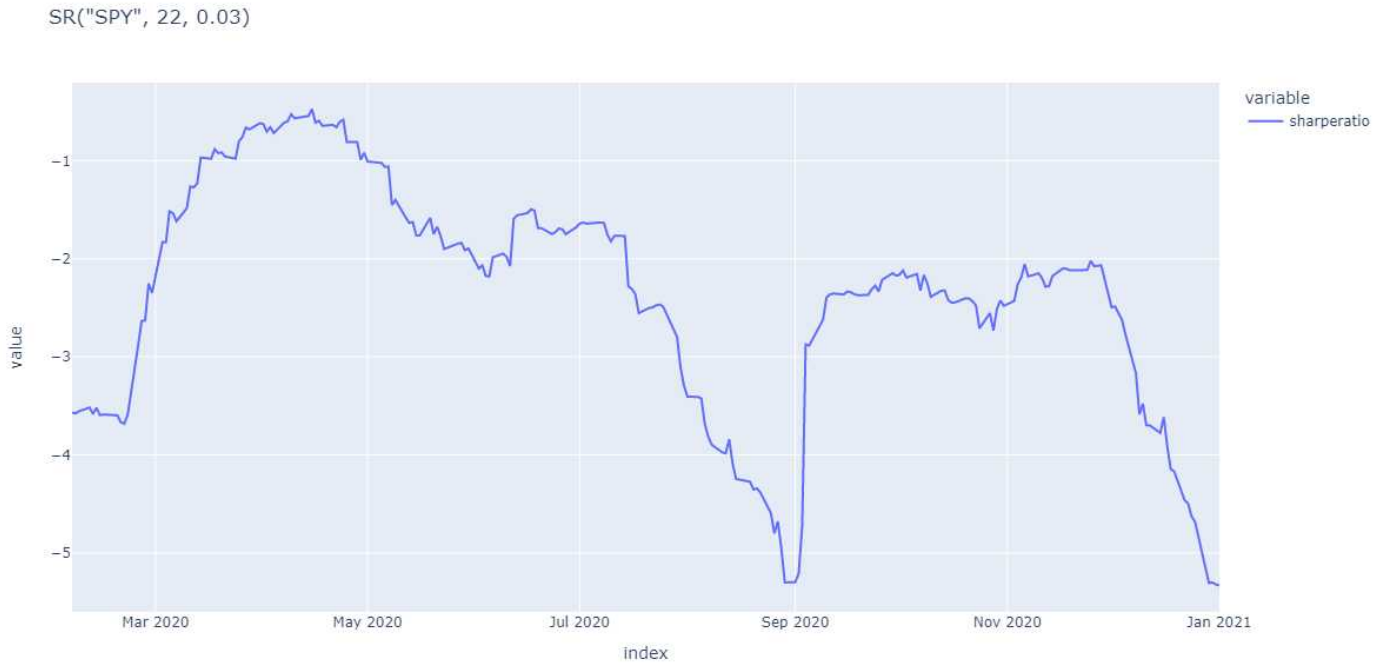
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class SharpeRatioAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SharpeRatio _sharperatio;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _sharperatio = new SharpeRatio(22, 0.03m);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _sharperatio.Update(bar.EndTime, bar.Close);

        if (_sharperatio.IsReady)
        {
            // The current value of _sharperatio is represented by itself (_sharperatio)
            // or _sharperatio.Current.Value
            Plot("SharpeRatio", "sharperatio", _sharperatio);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `SharpeRatio` indicator properties:



Indicator History

To get the historical data of the `SharpeRatio` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class SharpeRatioAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SharpeRatio _sr;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _sr = SR(_symbol, 22, 0.03m);

        var indicatorHistory = IndicatorHistory(_sr, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_sr, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_sr, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Simple Moving Average

Introduction

This indicator represents the traditional simple moving average indicator (SMA)

To view the implementation of this indicator, see the LEAN GitHub repository .

Using SMA Indicator

To create an automatic indicator for `SimpleMovingAverage` , call the `SMA` helper method from the `QCAlgorithm` class. The `SMA` method creates a `SimpleMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class SimpleMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SimpleMovingAverage _sma;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _sma = SMA(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_sma.IsReady)
        {
            // The current value of _sma is represented by itself (_sma)
            // or _sma.Current.Value
            Plot("SimpleMovingAverage", "sma", _sma);
            // Plot all properties of abands
            Plot("SimpleMovingAverage", "rollingsum", _sma.RollingSum);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `SimpleMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

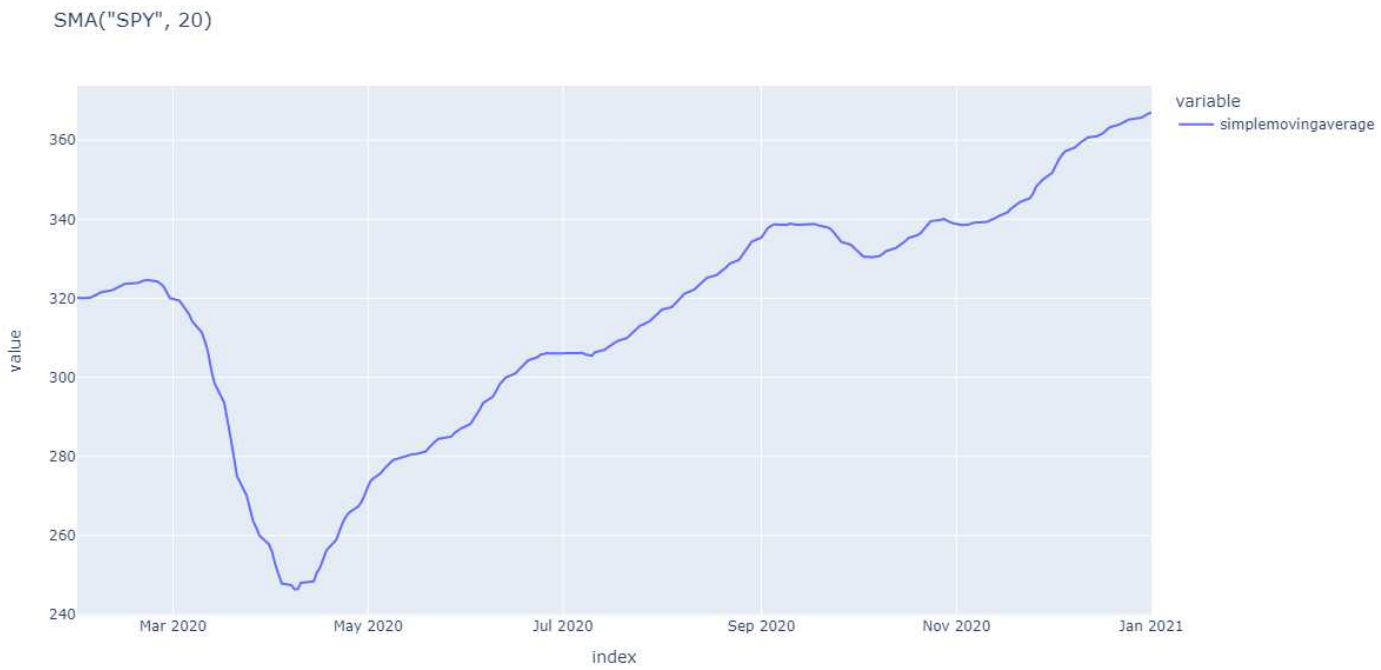
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class SimpleMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SimpleMovingAverage _simplemovingaverage;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _simplemovingaverage = new SimpleMovingAverage(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _simplemovingaverage.Update(bar.EndTime, bar.Close);

        if (_simplemovingaverage.IsReady)
        {
            // The current value of _simplemovingaverage is represented by itself (_simplemovingaverage)
            // or _simplemovingaverage.Current.Value
            Plot("SimpleMovingAverage", "simplemovingaverage", _simplemovingaverage);
            // Plot all properties of abands
            Plot("SimpleMovingAverage", "rollingsum", _simplemovingaverage.RollingSum);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `SimpleMovingAverage` indicator properties:



Indicator History

To get the historical data of the `SimpleMovingAverage` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class SimpleMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SimpleMovingAverage _sma;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _sma = SMA(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_sma, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_sma, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_sma, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var rollingSum = indicatorHistory.Select(x => ((dynamic)x).RollingSum).ToList();
    }
}
```

Supported Indicators

Smoothed On Balance Volume

Introduction

To view the implementation of this indicator, see the LEAN GitHub repository .

Using SOBV Indicator

To create an automatic indicator for `SmoothedOnBalanceVolume` , call the `SOBV` helper method from the `QCAlgorithm` class. The `SOBV` method creates a `SmoothedOnBalanceVolume` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class SmoothedOnBalanceVolumeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SmoothedOnBalanceVolume _sobv;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _sobv = SOBV(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_sobv.IsReady)
        {
            // The current value of _sobv is represented by itself (_sobv)
            // or _sobv.Current.Value
            Plot("SmoothedOnBalanceVolume", "sobv", _sobv);
            // Plot all properties of abands
            Plot("SmoothedOnBalanceVolume", "onbalancevolume", _sobv.OnBalanceVolume);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `SmoothedOnBalanceVolume` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

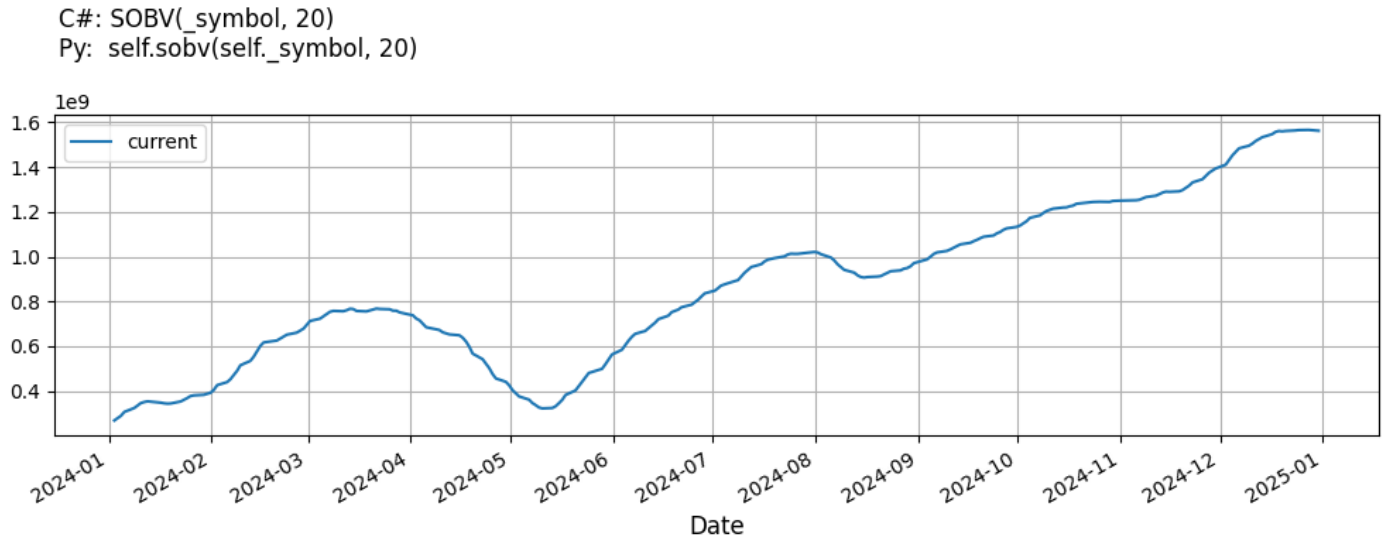
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class SmoothedOnBalanceVolumeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SmoothedOnBalanceVolume _smoothedonbalancevolume;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _smoothedonbalancevolume = new SmoothedOnBalanceVolume(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _smoothedonbalancevolume.Update(bar);

        if (_smoothedonbalancevolume.IsReady)
        {
            // The current value of _smoothedonbalancevolume is represented by itself (_smoothedonbalancevolume)
            // or _smoothedonbalancevolume.Current.Value
            Plot("SmoothedOnBalanceVolume", "smoothedonbalancevolume", _smoothedonbalancevolume);
            // Plot all properties of abands
            Plot("SmoothedOnBalanceVolume", "onbalancevolume", _smoothedonbalancevolume.OnBalanceVolume);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `SmoothedOnBalanceVolume` indicator properties:



Indicator History

To get the historical data of the `SmoothedOnBalanceVolume` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class SmoothedOnBalanceVolumeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SmoothedOnBalanceVolume _sobv;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _sobv = SOBV(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_sobv, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_sobv, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_sobv, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var onBalanceVolume = indicatorHistory.Select(x => ((dynamic)x).OnBalanceVolume).ToList();
    }
}
```

Supported Indicators

Sortino Ratio

Introduction

Calculation of the Sortino Ratio, a modification of the . Reference: https://www.cmegroup.com/education/files/rr-sortino-a-sharper-ratio.pdf Formula: S(x) = (R - T) / TDD Where: S(x) - Sortino ratio of x R - the average period return T - the target or required rate of return for the investment strategy under consideration. In Sortino's early work, T was originally known as the minimum acceptable return, or MAR. In his more recent work, MAR is now referred to as the Desired Target Return. TDD - the target downside deviation.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using SORTINO Indicator

To create an automatic indicator for `SortinoRatio` , call the `SORTINO` helper method from the `QCAlgorithm` class. The `SORTINO` method creates a `SortinoRatio` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

C#

```
public class SortinoRatioAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SortinoRatio _sortino;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _sortino = SORTINO(_symbol, 22);
    }

    public override void OnData(Slice data)
    {

        if (_sortino.IsReady)
        {
            // The current value of _sortino is represented by itself (_sortino)
            // or _sortino.Current.Value
            Plot("SortinoRatio", "sortino", _sortino);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `SortinoRatio` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

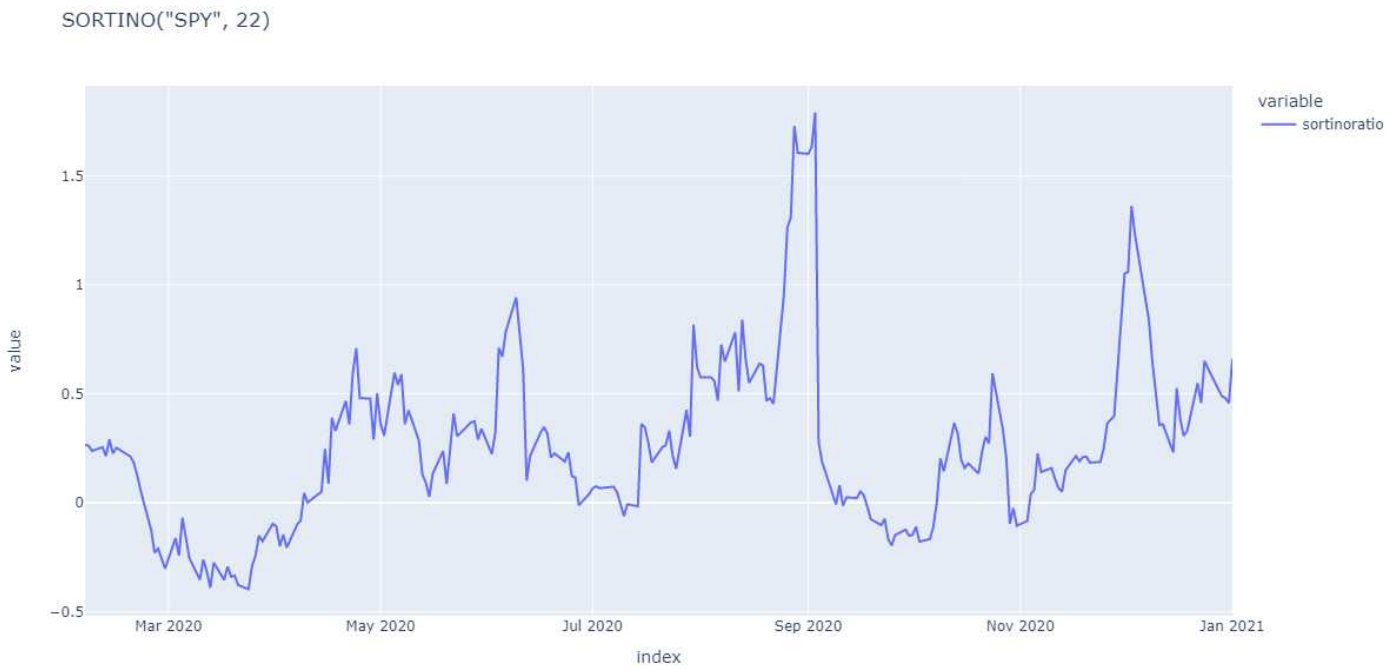
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class SortinoRatioAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SortinoRatio _sortinoratio;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _sortinoratio = new SortinoRatio(22);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _sortinoratio.Update(bar.EndTime, bar.Close);

        if (_sortinoratio.IsReady)
        {
            // The current value of _sortinoratio is represented by itself (_sortinoratio)
            // or _sortinoratio.Current.Value
            Plot("SortinoRatio", "sortinoratio", _sortinoratio);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `SortinoRatio` indicator properties:



Indicator History

To get the historical data of the `SortinoRatio` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class SortinoRatioAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SortinoRatio _sortino;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _sortino = SORTINO(_symbol, 22);

        var indicatorHistory = IndicatorHistory(_sortino, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_sortino, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_sortino, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Squeeze Momentum

Introduction

The SqueezeMomentum indicator calculates whether the market is in a "squeeze" condition, determined by comparing Bollinger Bands to Keltner Channels. When the Bollinger Bands are inside the Keltner Channels, the indicator returns 1 (squeeze on). Otherwise, it returns -1 (squeeze off).

To view the implementation of this indicator, see the LEAN GitHub repository .

Using SM Indicator

To create an automatic indicator for `SqueezeMomentum` , call the `SM` helper method from the `QCAlgorithm` class. The `SM` method creates a `SqueezeMomentum` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class SqueezeMomentumAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SqueezeMomentum _sm;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _sm = SM(_symbol, 20, 2, 20, 1.5m);
    }

    public override void OnData(Slice data)
    {

        if (_sm.IsReady)
        {
            // The current value of _sm is represented by itself (_sm)
            // or _sm.Current.Value
            Plot("SqueezeMomentum", "sm", _sm);
            // Plot all properties of abands
            Plot("SqueezeMomentum", "bollingerbands", _sm.BollingerBands);
            Plot("SqueezeMomentum", "keltnerchannels", _sm.KeltnerChannels);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `SqueezeMomentum` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

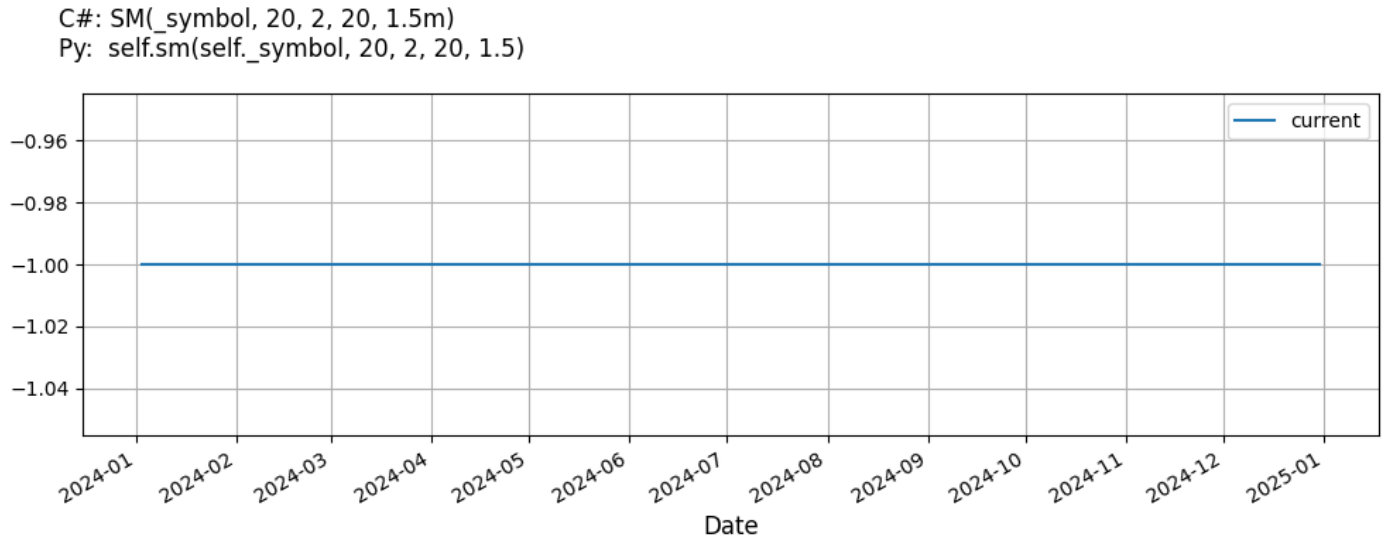
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class SqueezeMomentumAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SqueezeMomentum _squeezemomentum;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _squeezemomentum = new SqueezeMomentum("SPY", 20, 2m, 20, 1.5m);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _squeezemomentum.Update(bar);

        if (_squeezemomentum.IsReady)
        {
            // The current value of _squeezemomentum is represented by itself (_squeezemomentum)
            // or _squeezemomentum.Current.Value
            Plot("SqueezeMomentum", "squeezemomentum", _squeezemomentum);
            // Plot all properties of abands
            Plot("SqueezeMomentum", "bollingerbands", _squeezemomentum.BollingerBands);
            Plot("SqueezeMomentum", "keltnerchannels", _squeezemomentum.KeltnerChannels);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `SqueezeMomentum` indicator properties:



Indicator History

To get the historical data of the `SqueezeMomentum` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class SqueezeMomentumAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SqueezeMomentum _sm;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _sm = SM(_symbol, 20, 2, 20, 1.5m);

        var indicatorHistory = IndicatorHistory(_sm, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_sm, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_sm, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var bollingerBands = indicatorHistory.Select(x => ((dynamic)x).BollingerBands).ToList();
        var keltnerChannels = indicatorHistory.Select(x => ((dynamic)x).KeltnerChannels).ToList();
    }
}
```

Supported Indicators

Standard Deviation

Introduction

This indicator computes the n-period population standard deviation.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using STD Indicator

To create an automatic indicator for `StandardDeviation` , call the `STD` helper method from the `QCAlgorithm` class. The `STD` method creates a `StandardDeviation` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class StandardDeviationAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private StandardDeviation _std;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _std = STD(_symbol, 22);
    }

    public override void OnData(Slice data)
    {

        if (_std.IsReady)
        {
            // The current value of _std is represented by itself (_std)
            // or _std.Current.Value
            Plot("StandardDeviation", "std", _std);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `StandardDeviation` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

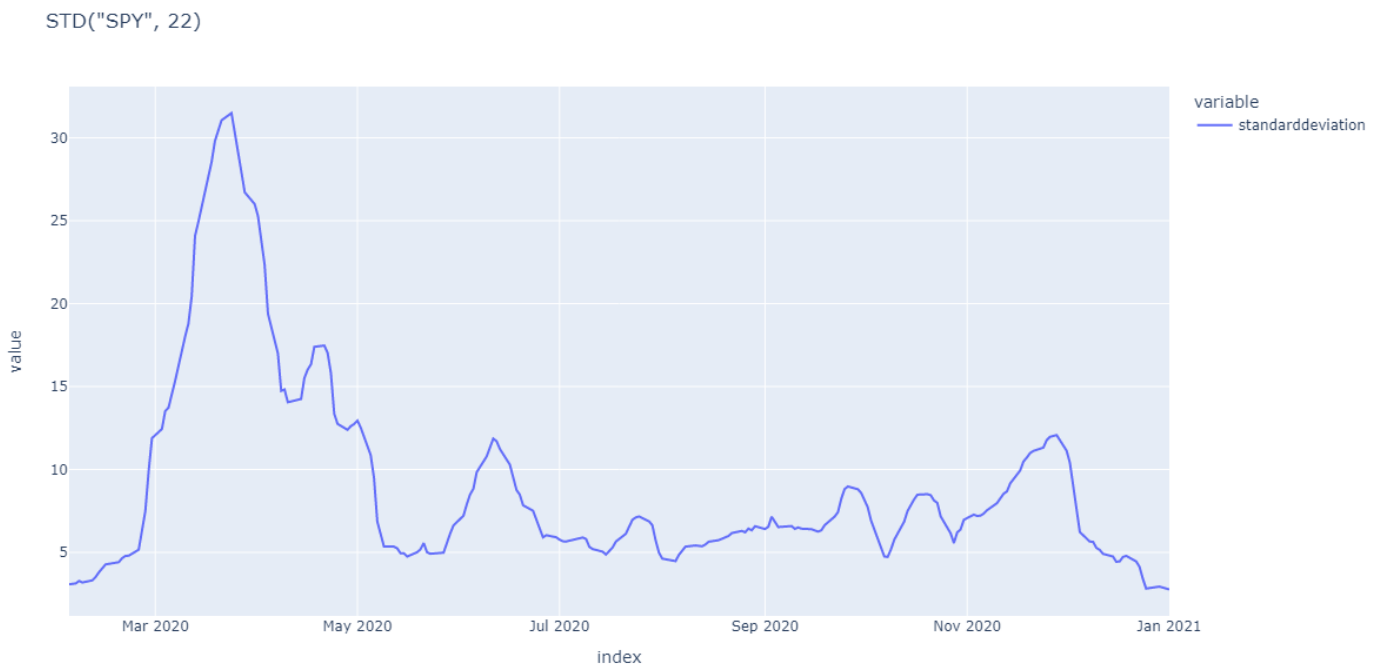
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class StandardDeviationAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private StandardDeviation _standarddeviation;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _standarddeviation = new StandardDeviation(22);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _standarddeviation.Update(bar.EndTime, bar.Close);

        if (_standarddeviation.IsReady)
        {
            // The current value of _standarddeviation is represented by itself (_standarddeviation)
            // or _standarddeviation.Current.Value
            Plot("StandardDeviation", "standarddeviation", _standarddeviation);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `StandardDeviation` indicator properties:



Indicator History

To get the historical data of the `StandardDeviation` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class StandardDeviationAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private StandardDeviation _std;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _std = STD(_symbol, 22);

        var indicatorHistory = IndicatorHistory(_std, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_std, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_std, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Stochastic

Introduction

This indicator computes the Slow Stochastics %K and %D. The Fast Stochastic %K is is computed by (Current Close Price - Lowest Price of given Period) / (Highest Price of given Period - Lowest Price of given Period) multiplied by 100. Once the Fast Stochastic %K is calculated the Slow Stochastic %K is calculated by the average/smoothed price of of the Fast %K with the given period. The Slow Stochastic %D is then derived from the Slow Stochastic %K with the given period.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using STO Indicator

To create an automatic indicator for `Stochastic` , call the `STO` helper method from the `QCAlgorithm` class. The `STO` method creates a `Stochastic` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class StochasticAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Stochastic _sto;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _sto = STO(_symbol, 20, 10, 20);
    }

    public override void OnData(Slice data)
    {

        if (_sto.IsReady)
        {
            // The current value of _sto is represented by itself (_sto)
            // or _sto.Current.Value
            Plot("Stochastic", "sto", _sto);
            // Plot all properties of abands
            Plot("Stochastic", "faststoch", _sto.FastStoch);
            Plot("Stochastic", "stochk", _sto.StochK);
            Plot("Stochastic", "stochd", _sto.StochD);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `Stochastic` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

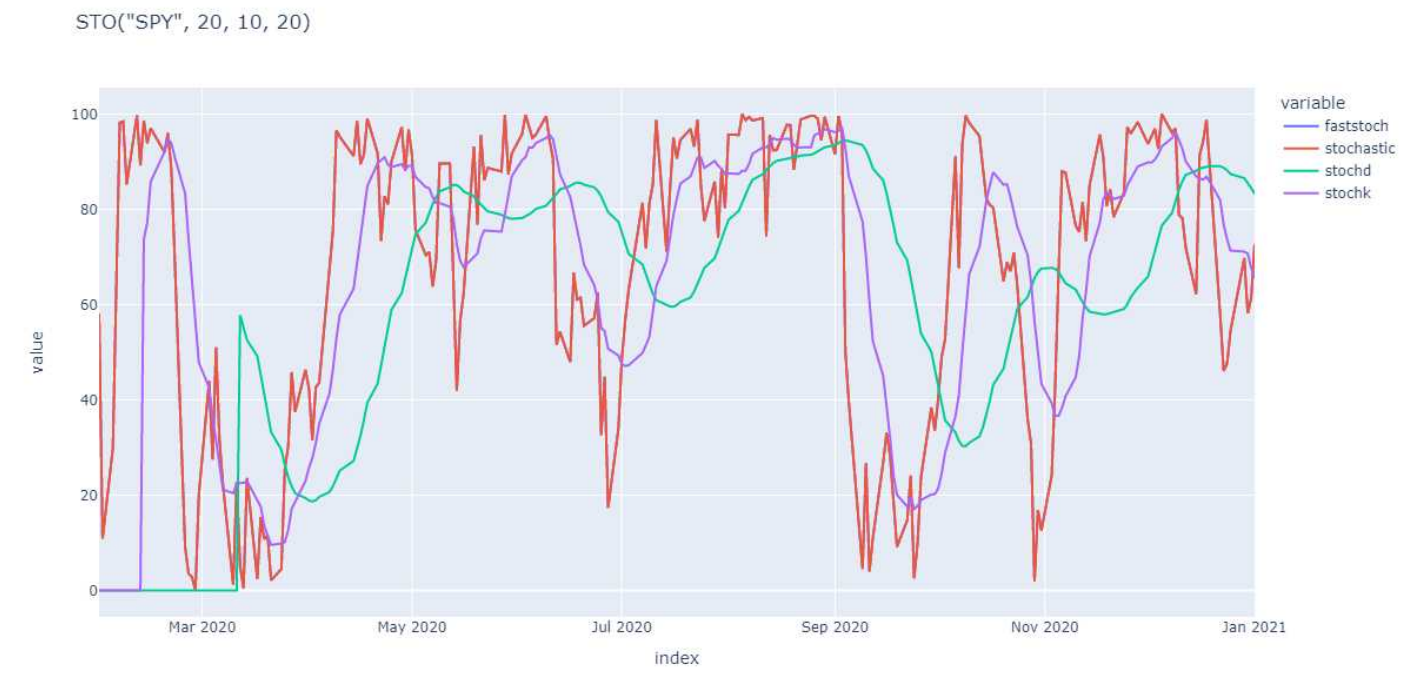
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class StochasticAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Stochastic _stochastic;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _stochastic = new Stochastic(20, 10, 20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _stochastic.Update(bar);

        if (_stochastic.IsReady)
        {
            // The current value of _stochastic is represented by itself (_stochastic)
            // or _stochastic.Current.Value
            Plot("Stochastic", "stochastic", _stochastic);
            // Plot all properties of abands
            Plot("Stochastic", "faststoch", _stochastic.FastStoch);
            Plot("Stochastic", "stochk", _stochastic.StochK);
            Plot("Stochastic", "stochd", _stochastic.StochD);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `Stochastic` indicator properties:



Indicator History

To get the historical data of the `Stochastic` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class StochasticAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Stochastic _sto;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _sto = STO(_symbol, 20, 10, 20);

        var indicatorHistory = IndicatorHistory(_sto, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_sto, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_sto, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var fastStoch = indicatorHistory.Select(x => ((dynamic)x).FastStoch).ToList();
        var stochK = indicatorHistory.Select(x => ((dynamic)x).StochK).ToList();
        var stochD = indicatorHistory.Select(x => ((dynamic)x).StochD).ToList();
    }
}
```

Supported Indicators

Stochastic Relative Strength Index

Introduction

Stochastic RSI, or simply StochRSI, is a technical analysis indicator used to determine whether an asset is overbought or oversold, as well as to identify current market trends. As the name suggests, the StochRSI is a derivative of the standard Relative Strength Index (RSI) and, as such, is considered an indicator of an indicator. It is a type of oscillator, meaning that it fluctuates above and below a center line.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using SRSI Indicator

To create an automatic indicator for `StochasticRelativeStrengthIndex` , call the `SRSI` helper method from the `QCAlgorithm` class. The `SRSI` method creates a `StochasticRelativeStrengthIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class StochasticRelativeStrengthIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private StochasticRelativeStrengthIndex _srsi;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _srsi = SRSI(_symbol, 14, 14, 3, 3);
    }

    public override void OnData(Slice data)
    {

        if (_srsi.IsReady)
        {
            // The current value of _srsi is represented by itself (_srsi)
            // or _srsi.Current.Value
            Plot("StochasticRelativeStrengthIndex", "srsi", _srsi);
            // Plot all properties of abands
            Plot("StochasticRelativeStrengthIndex", "k", _srsi.K);
            Plot("StochasticRelativeStrengthIndex", "d", _srsi.D);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `StochasticRelativeStrengthIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

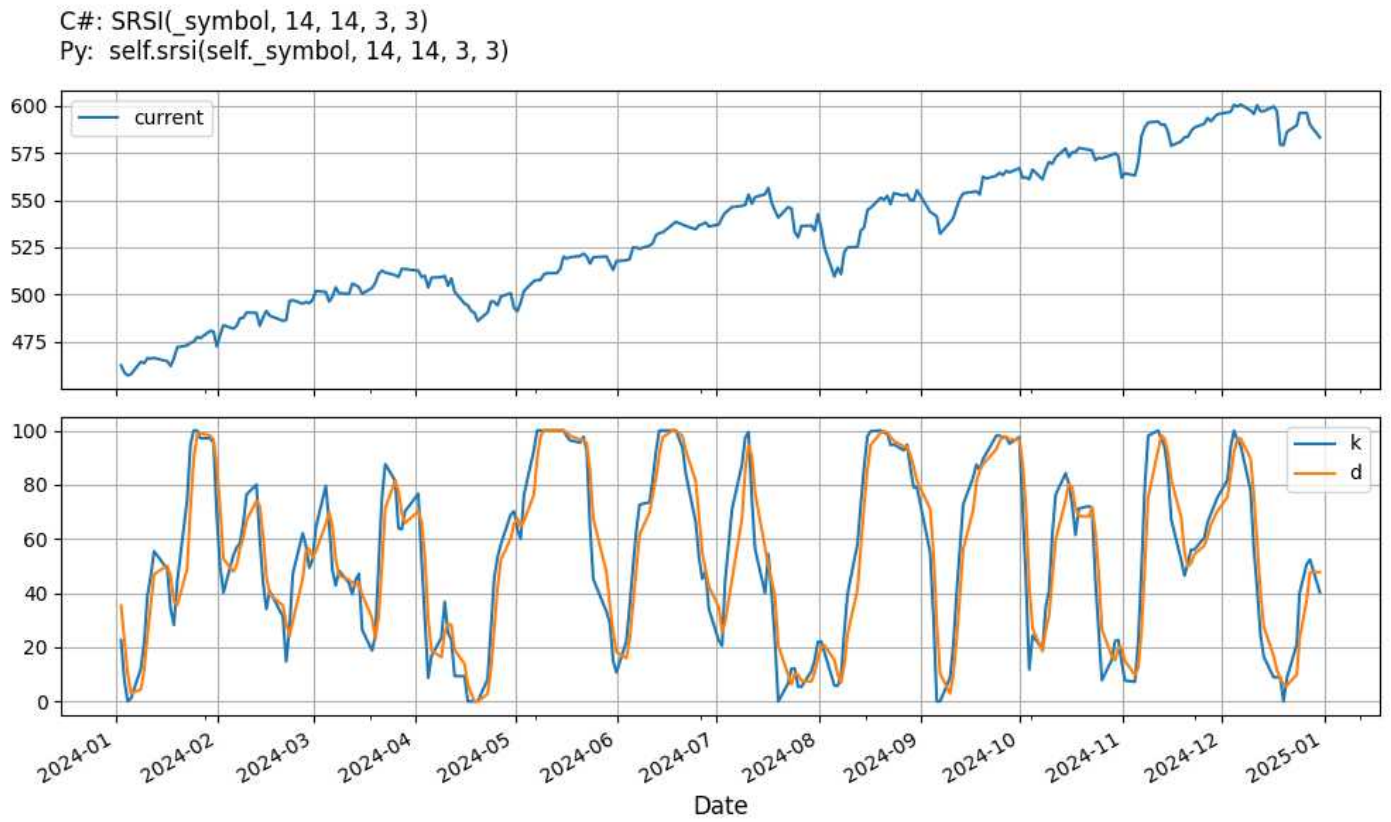
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class StochasticRelativeStrengthIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private StochasticRelativeStrengthIndex _stochasticrelativestrengthindex;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _stochasticrelativestrengthindex = new StochasticRelativeStrengthIndex(14, 14, 3, 3);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _stochasticrelativestrengthindex.Update(bar.EndTime, bar.Close);

        if (_stochasticrelativestrengthindex.IsReady)
        {
            // The current value of _stochasticrelativestrengthindex is represented by itself
(_stochasticrelativestrengthindex)
            // or _stochasticrelativestrengthindex.Current.Value
            Plot("StochasticRelativeStrengthIndex", "stochasticrelativestrengthindex",
_stochasticrelativestrengthindex);
            // Plot all properties of abands
            Plot("StochasticRelativeStrengthIndex", "k", _stochasticrelativestrengthindex.K);
            Plot("StochasticRelativeStrengthIndex", "d", _stochasticrelativestrengthindex.D);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `StochasticRelativeStrengthIndex` indicator properties:



Indicator History

To get the historical data of the `StochasticRelativeStrengthIndex` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history

requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
C#
public class StochasticRelativeStrengthIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private StochasticRelativeStrengthIndex _srsi;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _srsi = SRSI(_symbol, 14, 14, 3, 3);

        var indicatorHistory = IndicatorHistory(_srsi, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_srsi, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_srsi, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var k = indicatorHistory.Select(x => ((dynamic)x).K).ToList();
        var d = indicatorHistory.Select(x => ((dynamic)x).D).ToList();
    }
}
```

Supported Indicators

Sum

Introduction

This indicator represents an indicator capable of tracking the sum for the given period

To view the implementation of this indicator, see the LEAN GitHub repository .

Using SUM Indicator

To create an automatic indicator for `Sum` , call the `SUM` helper method from the `QCAlgorithm` class. The `SUM` method creates a `Sum` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class SumAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Sum _sum;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _sum = SUM(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_sum.IsReady)
        {
            // The current value of _sum is represented by itself (_sum)
            // or _sum.Current.Value
            Plot("Sum", "sum", _sum);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `Sum` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

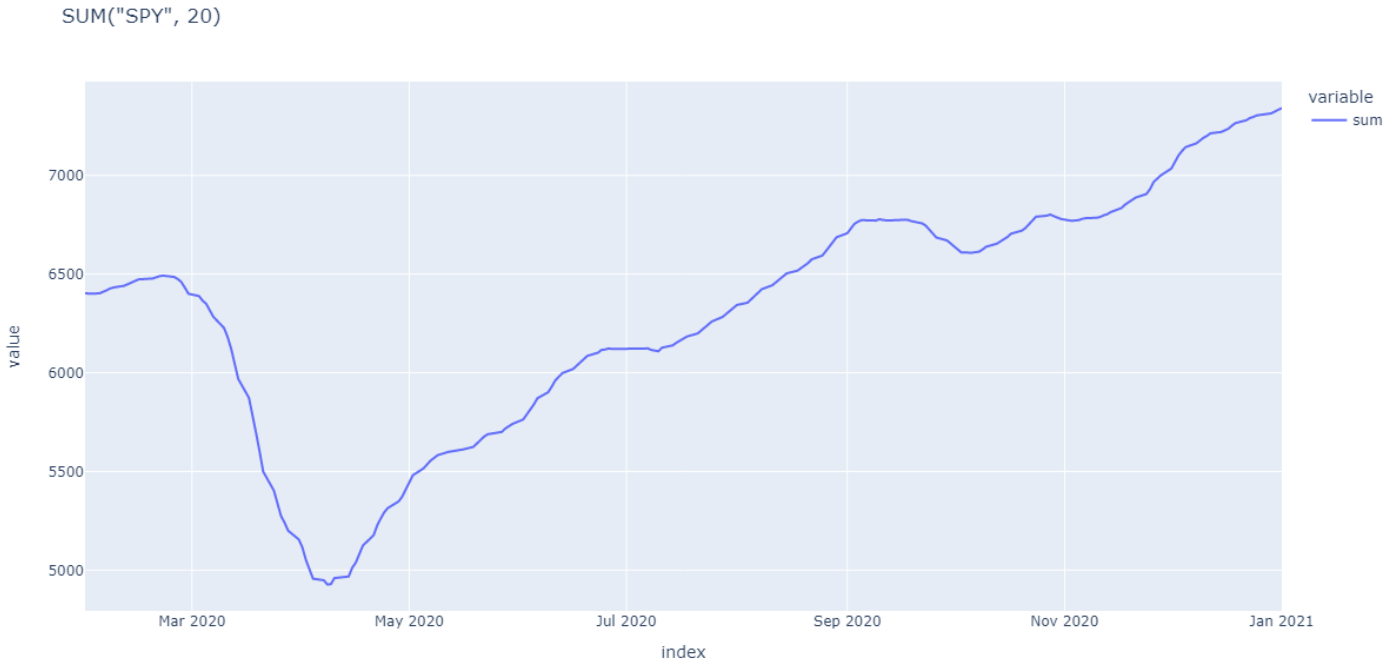
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class SumAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Sum _sum;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _sum = new Sum(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _sum.Update(bar.EndTime, bar.Close);

        if (_sum.IsReady)
        {
            // The current value of _sum is represented by itself (_sum)
            // or _sum.Current.Value
            Plot("Sum", "sum", _sum);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `Sum` indicator properties:



Indicator History

To get the historical data of the `Sum` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class SumAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Sum _sum;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _sum = SUM(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_sum, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_sum, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_sum, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Super Trend

Introduction

Super trend indicator. Formula can be found here via the excel file: https://tradingtuitions.com/supertrend-indicator-excel-sheet-with-realtime-buy-sell-signals/

To view the implementation of this indicator, see the LEAN GitHub repository .

Using STR Indicator

To create an automatic indicator for `SuperTrend` , call the `STR` helper method from the `QCAlgorithm` class. The `STR` method creates a `SuperTrend` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class SuperTrendAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SuperTrend _str;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _str = STR(_symbol, 20, 2, MovingAverageType.Wilders);
    }

    public override void OnData(Slice data)
    {

        if (_str.IsReady)
        {
            // The current value of _str is represented by itself (_str)
            // or _str.Current.Value
            Plot("SuperTrend", "str", _str);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `SuperTrend` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

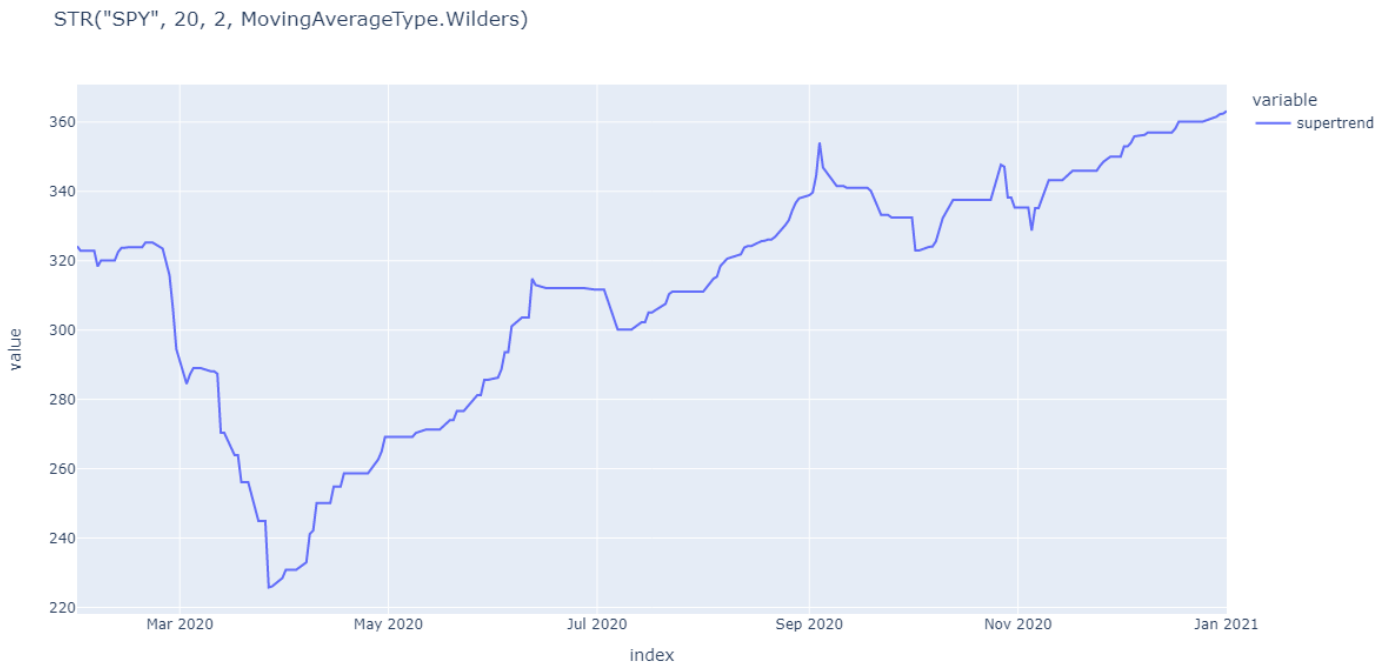
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class SuperTrendAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SuperTrend _supertrend;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _supertrend = new SuperTrend(20, 2, MovingAverageType.Wilders);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _supertrend.Update(bar);

        if (_supertrend.IsReady)
        {
            // The current value of _supertrend is represented by itself (_supertrend)
            // or _supertrend.Current.Value
            Plot("SuperTrend", "supertrend", _supertrend);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `SuperTrend` indicator properties:



Indicator History

To get the historical data of the `SuperTrend` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class SuperTrendAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SuperTrend _str;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _str = STR(_symbol, 20, 2, MovingAverageType.Wilders);

        var indicatorHistory = IndicatorHistory(_str, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_str, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_str, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Swiss Army Knife

Introduction

Swiss Army Knife indicator by John Ehlers

To view the implementation of this indicator, see the LEAN GitHub repository .

Using SWISS Indicator

To create an automatic indicator for `SwissArmyKnife` , call the `SWISS` helper method from the `QCAlgorithm` class. The `SWISS` method creates a `SwissArmyKnife` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class SwissArmyKnifeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SwissArmyKnife _swiss;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _swiss = SWISS(_symbol, 20, 0.2, SwissArmyKnifeTool.Gauss);
    }

    public override void OnData(Slice data)
    {

        if (_swiss.IsReady)
        {
            // The current value of _swiss is represented by itself (_swiss)
            // or _swiss.Current.Value
            Plot("SwissArmyKnife", "swiss", _swiss);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `SwissArmyKnife` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

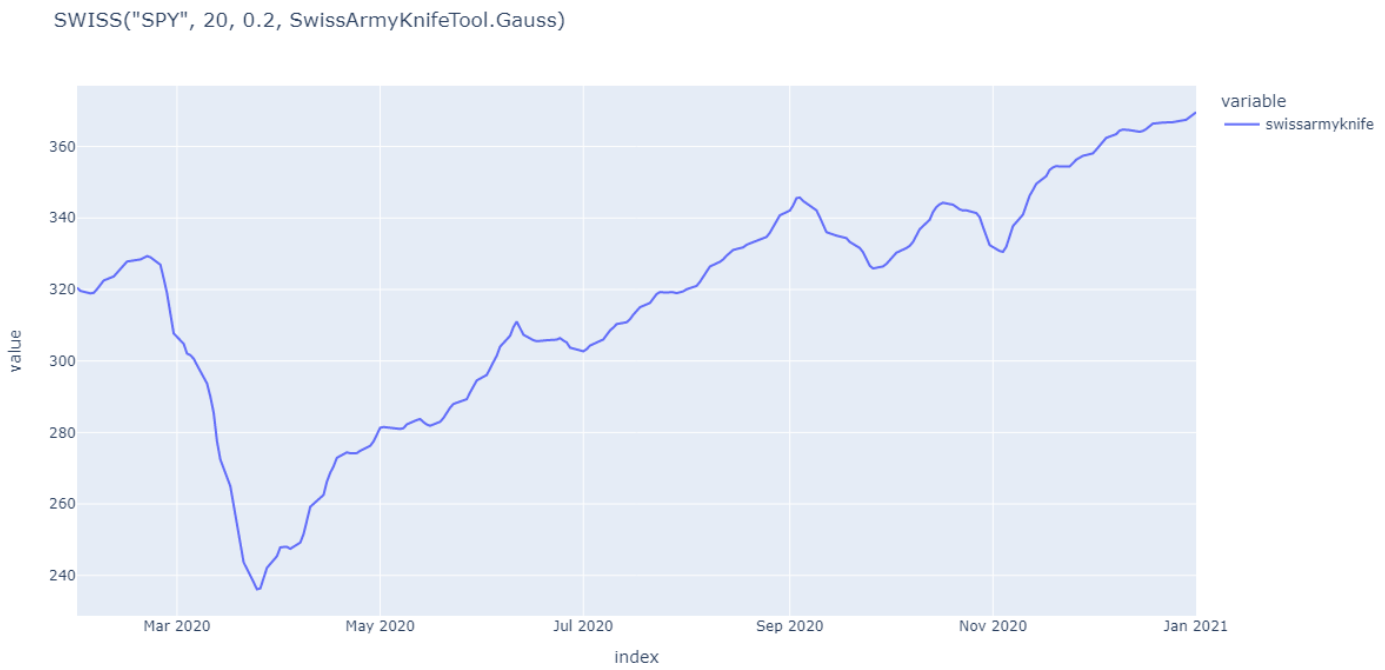
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class SwissArmyKnifeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SwissArmyKnife _swissarmyknife;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _swissarmyknife = new SwissArmyKnife(20, 0.2, SwissArmyKnifeTool.Gauss);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _swissarmyknife.Update(bar.EndTime, bar.Close);

        if (_swissarmyknife.IsReady)
        {
            // The current value of _swissarmyknife is represented by itself (_swissarmyknife)
            // or _swissarmyknife.Current.Value
            Plot("SwissArmyKnife", "swissarmyknife", _swissarmyknife);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `SwissArmyKnife` indicator properties:



Indicator History

To get the historical data of the `SwissArmyKnife` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class SwissArmyKnifeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private SwissArmyKnife _swiss;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _swiss = SWISS(_symbol, 20, 0.2, SwissArmyKnifeTool.Gauss);

        var indicatorHistory = IndicatorHistory(_swiss, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_swiss, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_swiss, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

T3 Moving Average

Introduction

This indicator computes the T3 Moving Average (T3). The T3 Moving Average is calculated with the following formula: EMA1(x, Period) = EMA(x, Period) EMA2(x, Period) = EMA(EMA1(x, Period),Period) GD(x, Period, volumeFactor) = (EMA1(x, Period)* (1+volumeFactor)) - (EMA2(x, Period)* volumeFactor) T3 = GD(GD(GD(t, Period, volumeFactor), Period, volumeFactor), Period, volumeFactor);

To view the implementation of this indicator, see the LEAN GitHub repository .

Using T3 Indicator

To create an automatic indicator for `T3MovingAverage` , call the `T3` helper method from the `QCAlgorithm` class. The `T3` method creates a `T3MovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

C#

```
public class T3MovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private T3MovingAverage _t3;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _t3 = T3(_symbol, 30, 0.7m);
    }

    public override void OnData(Slice data)
    {

        if (_t3.IsReady)
        {
            // The current value of _t3 is represented by itself (_t3)
            // or _t3.Current.Value
            Plot("T3MovingAverage", "t3", _t3);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `T3MovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

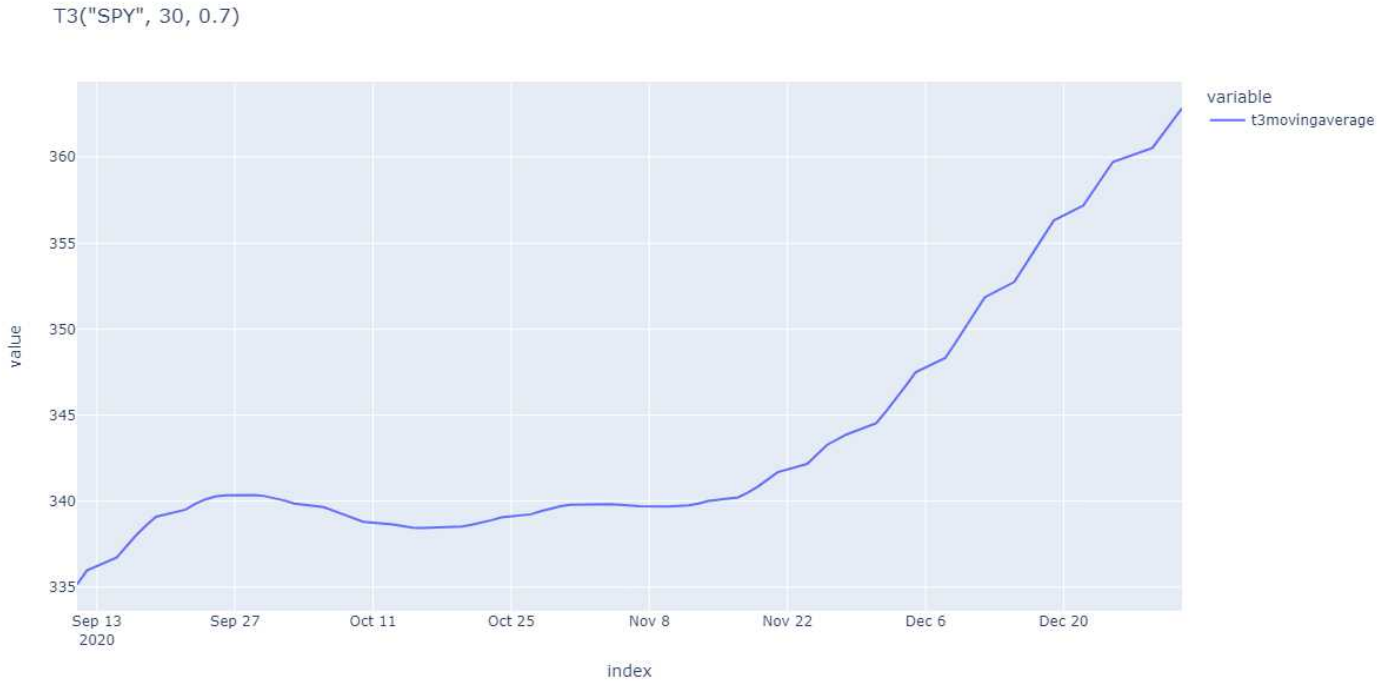
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class T3MovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private T3MovingAverage _t3movingaverage;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _t3movingaverage = new T3MovingAverage(30, 0.7m);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _t3movingaverage.Update(bar.EndTime, bar.Close);

        if (_t3movingaverage.IsReady)
        {
            // The current value of _t3movingaverage is represented by itself (_t3movingaverage)
            // or _t3movingaverage.Current.Value
            Plot("T3MovingAverage", "t3movingaverage", _t3movingaverage);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `T3MovingAverage` indicator properties:



Indicator History

To get the historical data of the `T3MovingAverage` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class T3MovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private T3MovingAverage _t3;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _t3 = T3(_symbol, 30, 0.7m);

        var indicatorHistory = IndicatorHistory(_t3, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_t3, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_t3, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Target Downside Deviation

Introduction

This indicator computes the n-period target downside deviation. The target downside deviation is defined as the root-mean-square, or RMS, of the deviations of the realized return's underperformance from the target return where all returns above the target return are treated as underperformance of 0. Reference: https://www.cmegroup.com/education/files/rr-sortino-a-sharper-ratio.pdf

To view the implementation of this indicator, see the LEAN GitHub repository .

Using TDD Indicator

To create an automatic indicator for `TargetDownsideDeviation` , call the `TDD` helper method from the `QCAlgorithm` class. The `TDD` method creates a `TargetDownsideDeviation` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class TargetDownsideDeviationAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TargetDownsideDeviation _tdd;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tdd = TDD(_symbol, 50);
    }

    public override void OnData(Slice data)
    {

        if (_tdd.IsReady)
        {
            // The current value of _tdd is represented by itself (_tdd)
            // or _tdd.Current.Value
            Plot("TargetDownsideDeviation", "tdd", _tdd);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `TargetDownsideDeviation` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

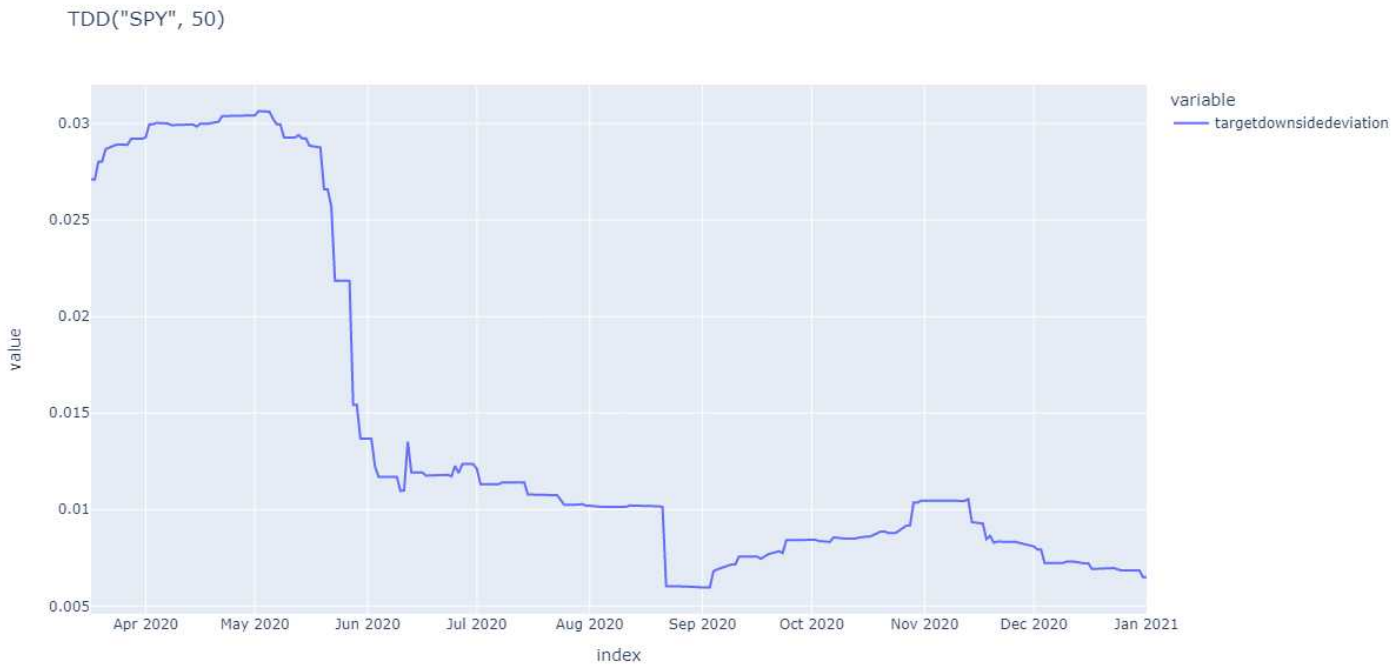
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class TargetDownsideDeviationAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TargetDownsideDeviation _targetdownsidedeviation;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _targetdownsidedeviation = new TargetDownsideDeviation(50).Of(new RateOfChange(1));
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _targetdownsidedeviation.Update(bar.EndTime, bar.Close);

        if (_targetdownsidedeviation.IsReady)
        {
            // The current value of _targetdownsidedeviation is represented by itself (_targetdownsidedeviation)
            // or _targetdownsidedeviation.Current.Value
            Plot("TargetDownsideDeviation", "targetdownsidedeviation", _targetdownsidedeviation);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `TargetDownsideDeviation` indicator properties:



Indicator History

To get the historical data of the `TargetDownsideDeviation` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class TargetDownsideDeviationAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TargetDownsideDeviation _tdd;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tdd = TDD(_symbol, 50);

        var indicatorHistory = IndicatorHistory(_tdd, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_tdd, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_tdd, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Theta

Introduction

Option Theta indicator that calculate the theta of an option

To view the implementation of this indicator, see the LEAN GitHub repository .

Using T Indicator

To create an automatic indicator for `Theta` , call the `T` helper method from the `QCAlgorithm` class. The `T` method creates a `Theta` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class ThetaAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _option, _mirrorOption;
    private Theta _t;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 20);

        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _option = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920C00564000");
        _mirrorOption = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920P00564000");
        AddOptionContract(_option, Resolution.Daily);
        AddOptionContract(_mirrorOption, Resolution.Daily);

        _t = T(_option, _mirrorOption);
    }

    public override void OnData(Slice data)
    {
        if (_t.IsReady)
        {
            // The current value of _t is represented by itself (_t)
            // or _t.Current.Value
            Plot("Theta", "t", _t);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `Theta` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

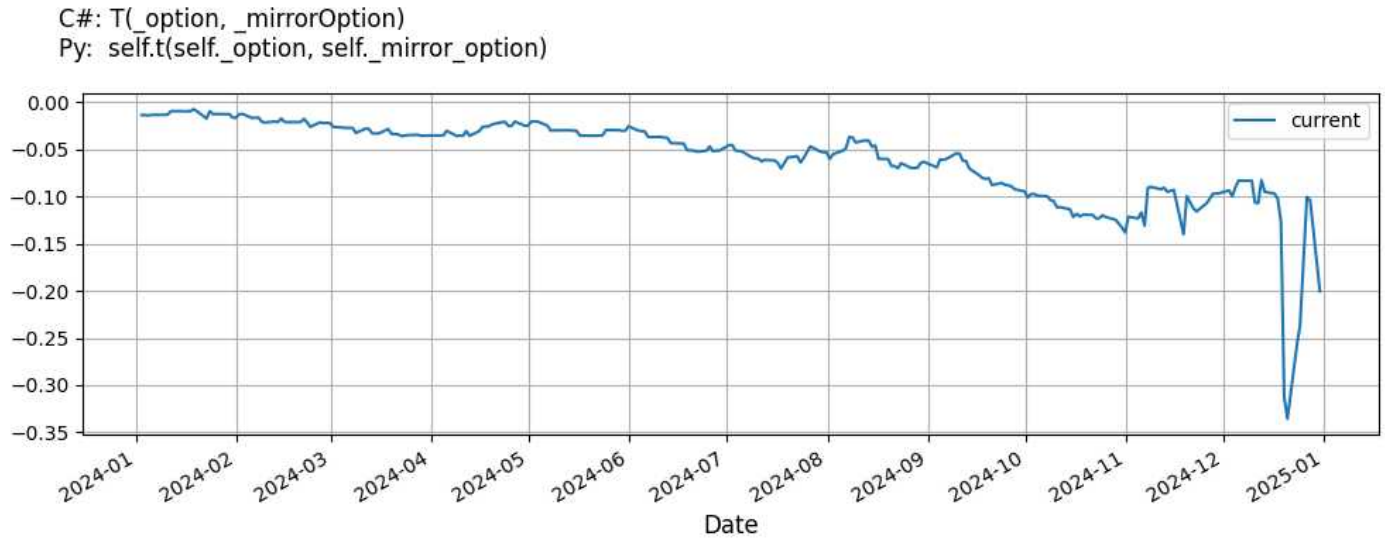
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ThetaAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _option, _mirrorOption;
    private Theta _theta;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 20);

        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        var interestRateModel = new InterestRateProvider();
        var dividendYieldModel = new DividendYieldProvider(_symbol);

        _option = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920C00564000");
        _mirrorOption = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920P00564000");
        AddOptionContract(_option, Resolution.Daily);
        AddOptionContract(_mirrorOption, Resolution.Daily);

        _theta = new Theta(_option, interestRateModel, dividendYieldModel, _mirrorOption);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _theta.Update(new IndicatorDataPoint(_symbol, bar.EndTime, bar.Close));
        if (data.QuoteBars.TryGetValue(_option, out var quoteBar))
            _theta.Update(new IndicatorDataPoint(_option, quoteBar.EndTime, quoteBar.Close));
        if (data.QuoteBars.TryGetValue(_mirrorOption, out quoteBar))
            _theta.Update(new IndicatorDataPoint(_mirrorOption, quoteBar.EndTime, quoteBar.Close));

        if (_theta.IsReady)
        {
            // The current value of _theta is represented by itself (_theta)
            // or _theta.Current.Value
            Plot("Theta", "theta", _theta);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `Theta` indicator properties:



Indicator History

To get the historical data of the `Theta` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory`

method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#

public class ThetaAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _option, _mirrorOption;
    private Theta _t;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 20);

        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _option = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920C00564000");
        _mirrorOption = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920P00564000");
        AddOptionContract(_option, Resolution.Daily);
        AddOptionContract(_mirrorOption, Resolution.Daily);

        _t = T(_option, _mirrorOption);

        var indicatorHistory = IndicatorHistory(_t, new[] { _symbol, _option, _mirrorOption }, 100,
Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_t, new[] { _symbol, _option, _mirrorOption },
TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_t, new[] { _symbol, _option, _mirrorOption }, new
DateTime(2024, 7, 1), new DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Time Profile

Introduction

This indicator represents an Indicator of the Market Profile with Time Price Opportunity (TPO) mode and its attributes

To view the implementation of this indicator, see the LEAN GitHub repository .

Using TP Indicator

To create an automatic indicator for `TimeProfile` , call the `TP` helper method from the `QCAlgorithm` class. The `TP` method creates a `TimeProfile` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class TimeProfileAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TimeProfile _tp;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tp = TP(_symbol, 3, 0.70m, 0.05m);
    }

    public override void OnData(Slice data)
    {

        if (_tp.IsReady)
        {
            // The current value of _tp is represented by itself (_tp)
            // or _tp.Current.Value
            Plot("TimeProfile", "tp", _tp);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `TimeProfile` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class TimeProfileAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TimeProfile _timeprofile;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _timeprofile = new TimeProfile(3, 0.70m, 0.05m);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _timeprofile.Update(bar);

        if (_timeprofile.IsReady)
        {
            // The current value of _timeprofile is represented by itself (_timeprofile)
            // or _timeprofile.Current.Value
            Plot("TimeProfile", "timeprofile", _timeprofile);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `TimeProfile` indicator properties:

Indicator History

To get the historical data of the `TimeProfile` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class TimeProfileAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TimeProfile _tp;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tp = TP(_symbol, 3, 0.70m, 0.05m);

        var indicatorHistory = IndicatorHistory(_tp, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_tp, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_tp, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Time Series Forecast

Introduction

This indicator represents an indicator capable of predicting new values given previous data from a window. source

To view the implementation of this indicator, see the LEAN GitHub repository .

Using TSF Indicator

To create an automatic indicator for `TimeSeriesForecast` , call the `TSF` helper method from the `QCAlgorithm` class. The `TSF` method creates a `TimeSeriesForecast` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class TimeSeriesForecastAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TimeSeriesForecast _tsf;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tsf = TSF(_symbol, 3);
    }

    public override void OnData(Slice data)
    {

        if (_tsf.IsReady)
        {
            // The current value of _tsf is represented by itself (_tsf)
            // or _tsf.Current.Value
            Plot("TimeSeriesForecast", "tsf", _tsf);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `TimeSeriesForecast` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

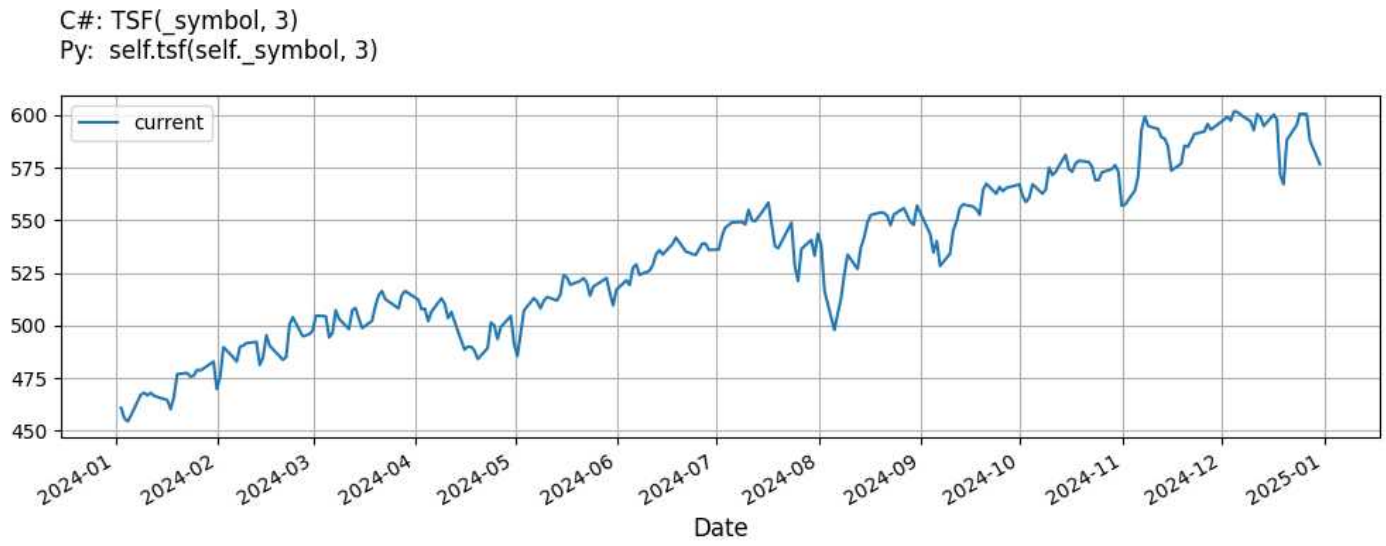
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class TimeSeriesForecastAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TimeSeriesForecast _timeseriesforecast;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _timeseriesforecast = new TimeSeriesForecast(3);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _timeseriesforecast.Update(bar.EndTime, bar.Close);

        if (_timeseriesforecast.IsReady)
        {
            // The current value of _timeseriesforecast is represented by itself (_timeseriesforecast)
            // or _timeseriesforecast.Current.Value
            Plot("TimeSeriesForecast", "timeseriesforecast", _timeseriesforecast);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `TimeSeriesForecast` indicator properties:



Indicator History

To get the historical data of the `TimeSeriesForecast` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class TimeSeriesForecastAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TimeSeriesForecast _tsf;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tsf = TSF(_symbol, 3);

        var indicatorHistory = IndicatorHistory(_tsf, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_tsf, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_tsf, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Tom Demark Sequential

Introduction

This indicator represents the Tom Demark Sequential indicator, which is used to identify potential trend exhaustion points. This implementation tracks both the setup and countdown phases, and can be extended to handle bullish and bearish setups, as well as qualifiers such as Perfect Setups and Countdown completions. - **Setup Phase**: Detects a trend by counting 9 consecutive bars where the close is less than (Buy Setup) or greater than (Sell Setup) the close 4 bars earlier. - **TDST Support/Resistance**: After a valid 9-bar setup completes, the indicator records the **lowest low** (for Sell Setup) or **highest high** (for Buy Setup) among the 9 bars. These are referred to as Support Price(for Buy Setup) and Resistance Price (for Sell Setup), and serve as critical thresholds that validate the continuation of the countdown phase. - **Countdown Phase**: Once a valid setup is completed, the indicator attempts to count 13 qualifying bars (not necessarily consecutive) where the close is less than (Buy Countdown) or greater than (Sell Countdown) the low/high 2 bars earlier. During this phase, the TDST Support/Resistance levels are checked — if the price breaks these levels, the countdown phase is reset, as the trend reversal condition is considered invalidated.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using TDS Indicator

To create an automatic indicator for `TomDemarkSequential` , call the `TDS` helper method from the `QCAlgorithm` class. The `TDS` method creates a `TomDemarkSequential` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class TomDemarkSequentialAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TomDemarkSequential _tds;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tds = TDS(_symbol);
    }

    public override void OnData(Slice data)
    {

        if (_tds.IsReady)
        {
            // The current value of _tds is represented by itself (_tds)
            // or _tds.Current.Value
            Plot("TomDemarkSequential", "tds", _tds);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `TomDemarkSequential` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To

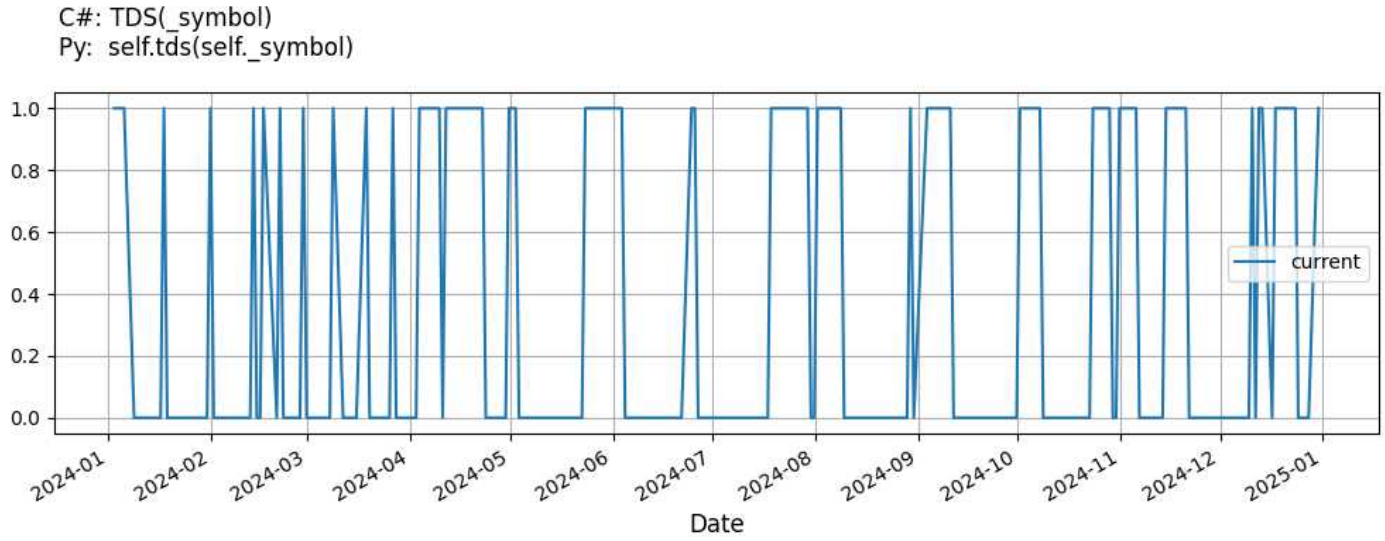
manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class TomDemarkSequentialAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TomDemarkSequential _tomdemarksequential;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tomdemarksequential = new TomDemarkSequential();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _tomdemarksequential.Update(bar);

        if (_tomdemarksequential.IsReady)
        {
            // The current value of _tomdemarksequential is represented by itself (_tomdemarksequential)
            // or _tomdemarksequential.Current.Value
            Plot("TomDemarkSequential", "tomdemarksequential", _tomdemarksequential);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `TomDemarkSequential` indicator properties:



Indicator History

To get the historical data of the `TomDemarkSequential` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class TomDemarkSequentialAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TomDemarkSequential _tds;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tds = TDS(_symbol);

        var indicatorHistory = IndicatorHistory(_tds, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_tds, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_tds, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Triangular Moving Average

Introduction

This indicator computes the Triangular Moving Average (TRIMA). The Triangular Moving Average is calculated with the following formula: (1) When the period is even, TRIMA(x,period)=SMA(SMA(x,period/2),(period/2)+1) (2) When the period is odd, TRIMA(x,period)=SMA(SMA(x,(period+1)/2),(period+1)/2)

To view the implementation of this indicator, see the LEAN GitHub repository .

Using TRIMA Indicator

To create an automatic indicator for `TriangularMovingAverage` , call the `TRIMA` helper method from the `QCAlgorithm` class. The `TRIMA` method creates a `TriangularMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class TriangularMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TriangularMovingAverage _trima;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _trima = TRIMA(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_trima.IsReady)
        {
            // The current value of _trima is represented by itself (_trima)
            // or _trima.Current.Value
            Plot("TriangularMovingAverage", "trima", _trima);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `TriangularMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

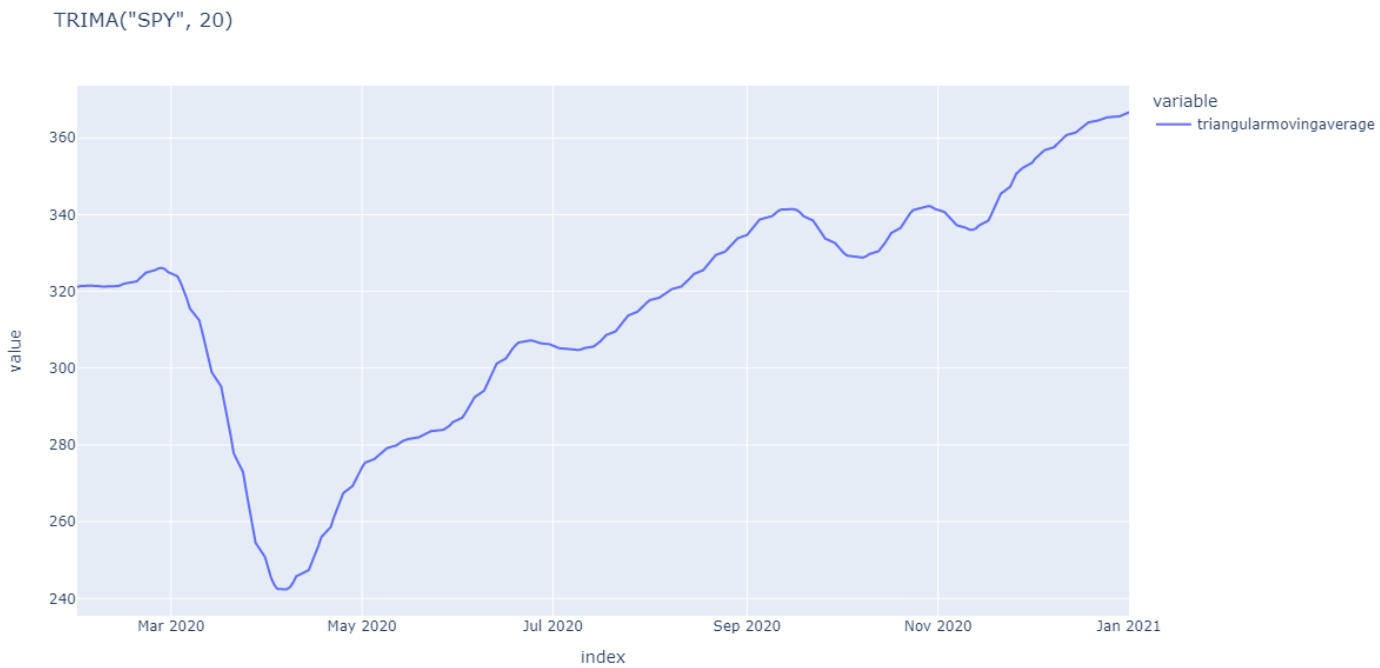
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class TriangularMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TriangularMovingAverage _triangularmovingaverage;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _triangularmovingaverage = new TriangularMovingAverage(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _triangularmovingaverage.Update(bar.EndTime, bar.Close);

        if (_triangularmovingaverage.IsReady)
        {
            // The current value of _triangularmovingaverage is represented by itself (_triangularmovingaverage)
            // or _triangularmovingaverage.Current.Value
            Plot("TriangularMovingAverage", "triangularmovingaverage", _triangularmovingaverage);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `TriangularMovingAverage` indicator properties:



Indicator History

To get the historical data of the `TriangularMovingAverage` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class TriangularMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TriangularMovingAverage _trima;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _trima = TRIMA(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_trima, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_trima, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_trima, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Triple Exponential Moving Average

Introduction

This indicator computes the Triple Exponential Moving Average (TEMA). The Triple Exponential Moving Average is calculated with the following formula: EMA1 = EMA(t,period) EMA2 = EMA(EMA(t,period),period) EMA3 = EMA(EMA(EMA(t,period),period),period) TEMA = 3 * EMA1 - 3 * EMA2 + EMA3

To view the implementation of this indicator, see the LEAN GitHub repository .

Using TEMA Indicator

To create an automatic indicator for `TripleExponentialMovingAverage` , call the `TEMA` helper method from the `QCAlgorithm` class. The `TEMA` method creates a `TripleExponentialMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class TripleExponentialMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TripleExponentialMovingAverage _tema;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tema = TEMA(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_tema.IsReady)
        {
            // The current value of _tema is represented by itself (_tema)
            // or _tema.Current.Value
            Plot("TripleExponentialMovingAverage", "tema", _tema);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `TripleExponentialMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class TripleExponentialMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TripleExponentialMovingAverage _tripleexponentialmovingaverage;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tripleexponentialmovingaverage = new TripleExponentialMovingAverage(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _tripleexponentialmovingaverage.Update(bar.EndTime, bar.Close);

        if (_tripleexponentialmovingaverage.IsReady)
        {
            // The current value of _tripleexponentialmovingaverage is represented by itself
(_tripleexponentialmovingaverage)
            // or _tripleexponentialmovingaverage.Current.Value
            Plot("TripleExponentialMovingAverage", "tripleexponentialmovingaverage",
_tripleexponentialmovingaverage);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `TripleExponentialMovingAverage` indicator properties:

Indicator History

To get the historical data of the `TripleExponentialMovingAverage` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class TripleExponentialMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TripleExponentialMovingAverage _tema;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tema = TEMA(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_tema, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_tema, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_tema, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Trix

Introduction

This indicator computes the TRIX (1-period ROC of a Triple EMA) The TRIX is calculated as explained here: http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:trix

To view the implementation of this indicator, see the LEAN GitHub repository .

Using TRIX Indicator

To create an automatic indicator for `Trix` , call the `TRIX` helper method from the `QCAlgorithm` class. The `TRIX` method creates a `Trix` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class TrixAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Trix _trix;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _trix = TRIX(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_trix.IsReady)
        {
            // The current value of _trix is represented by itself (_trix)
            // or _trix.Current.Value
            Plot("Trix", "trix", _trix);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `Trix` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

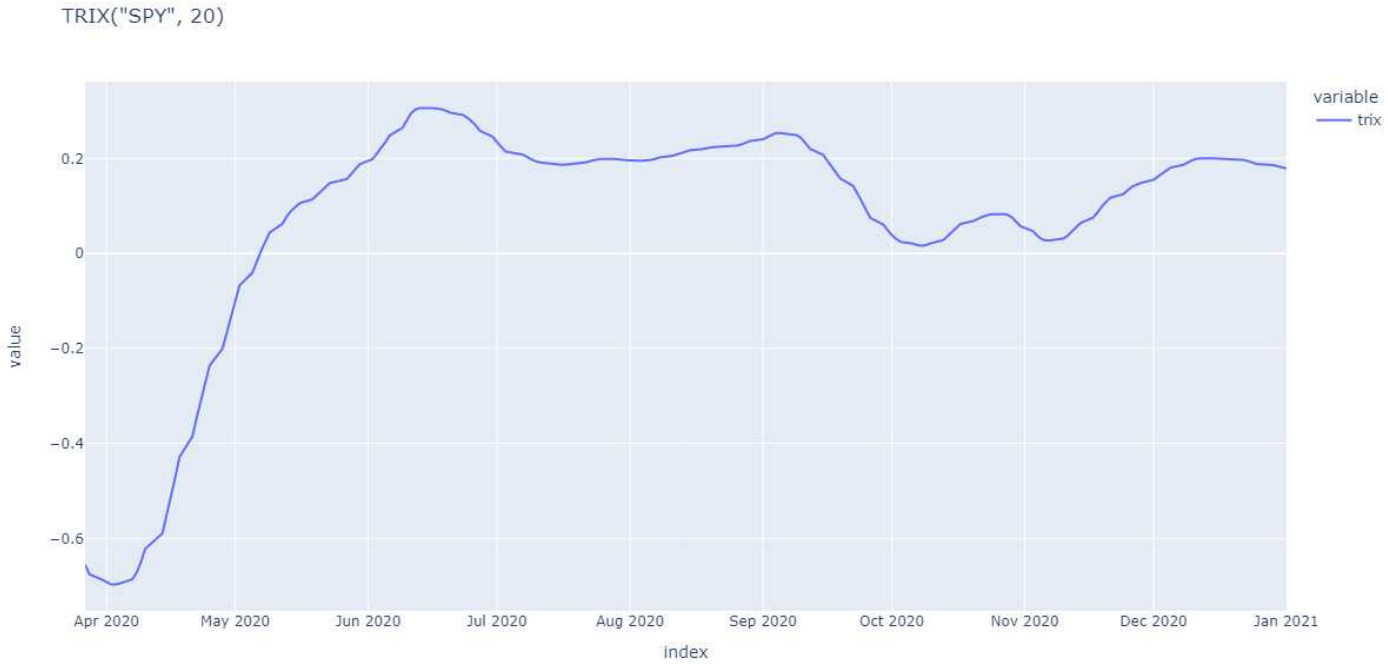
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class TrixAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Trix _trix;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _trix = new Trix(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _trix.Update(bar.EndTime, bar.Close);

        if (_trix.IsReady)
        {
            // The current value of _trix is represented by itself (_trix)
            // or _trix.Current.Value
            Plot("Trix", "trix", _trix);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `Trix` indicator properties:



Indicator History

To get the historical data of the `Trix` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class TrixAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Trix _trix;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _trix = TRIX(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_trix, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_trix, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_trix, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

True Range

Introduction

This indicator computes the True Range (TR). The True Range is the greatest of the following values: value1 = distance from today's high to today's low. value2 = distance from yesterday's close to today's high. value3 = distance from yesterday's close to today's low.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using TR Indicator

To create an automatic indicator for `TrueRange` , call the `TR` helper method from the `QCAlgorithm` class. The `TR` method creates a `TrueRange` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class TrueRangeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TrueRange _tr;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tr = TR(_symbol);
    }

    public override void OnData(Slice data)
    {

        if (_tr.IsReady)
        {
            // The current value of _tr is represented by itself (_tr)
            // or _tr.Current.Value
            Plot("TrueRange", "tr", _tr);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `TrueRange` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

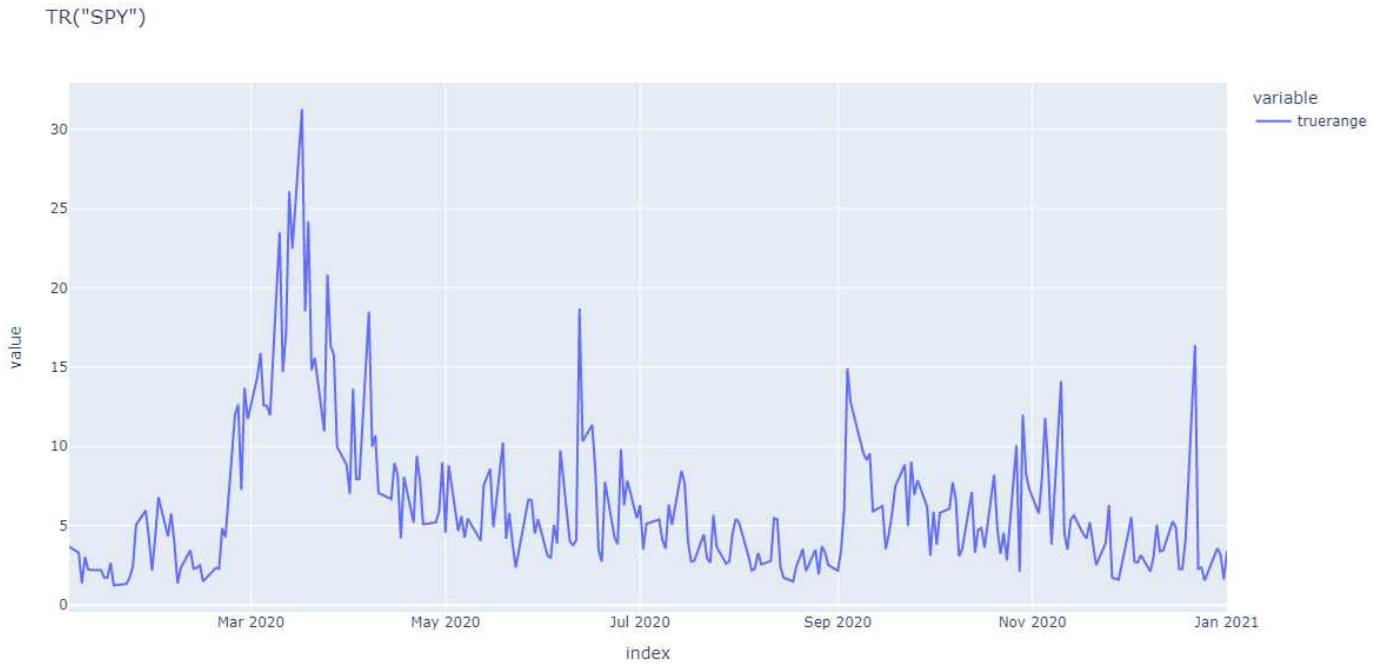
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class TrueRangeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TrueRange _truerange;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _truerange = new TrueRange();
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _truerange.Update(bar);

        if (_truerange.IsReady)
        {
            // The current value of _truerange is represented by itself (_truerange)
            // or _truerange.Current.Value
            Plot("TrueRange", "truerange", _truerange);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `TrueRange` indicator properties:



Indicator History

To get the historical data of the `TrueRange` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class TrueRangeAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TrueRange _tr;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tr = TR(_symbol);

        var indicatorHistory = IndicatorHistory(_tr, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_tr, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_tr, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

True Strength Index

Introduction

This indicator computes the True Strength Index (TSI). The True Strength Index is calculated as explained here: https://school.stockcharts.com/doku.php?id=technical_indicators:True_strength_index Briefly, the calculation has three steps: 1. Smooth the momentum and the absolute momentum by getting an EMA of them (typically of period 25) 2. Double smooth the momentum and the absolute momentum by getting an EMA of their EMA (typically of period 13) 3. The TSI formula itself: divide the double-smoothed momentum over the double-smoothed absolute momentum and multiply by 100 The signal is typically a 7-to-12-EMA of the TSI.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using TSI Indicator

To create an automatic indicator for `TrueStrengthIndex` , call the `TSI` helper method from the `QCAlgorithm` class. The `TSI` method creates a `TrueStrengthIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class TrueStrengthIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TrueStrengthIndex _tsi;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tsi = TSI(_symbol, 25, 13, 7, MovingAverageType.Exponential);
    }

    public override void OnData(Slice data)
    {

        if (_tsi.IsReady)
        {
            // The current value of _tsi is represented by itself (_tsi)
            // or _tsi.Current.Value
            Plot("TrueStrengthIndex", "tsi", _tsi);
            // Plot all properties of abands
            Plot("TrueStrengthIndex", "signal", _tsi.Signal);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `TrueStrengthIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

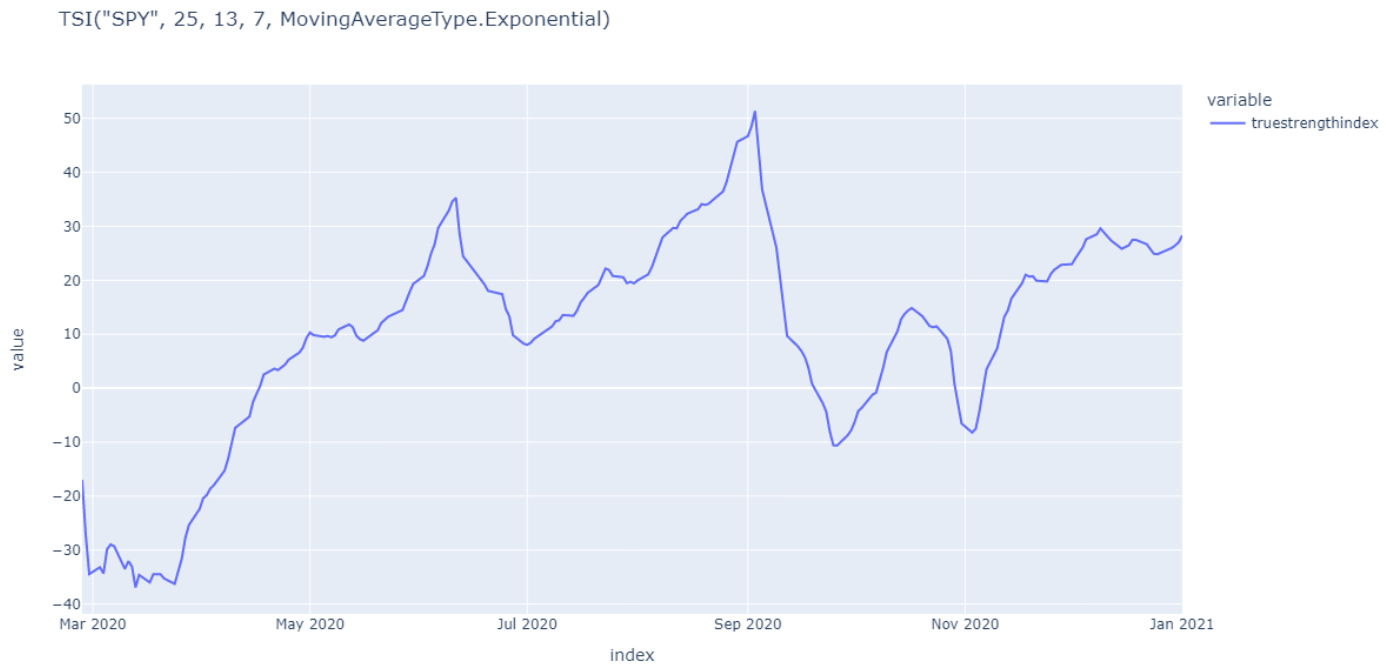
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class TrueStrengthIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TrueStrengthIndex _truestrengthindex;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _truestrengthindex = new TrueStrengthIndex(25, 13, 7, MovingAverageType.Exponential);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _truestrengthindex.Update(bar.EndTime, bar.Close);

        if (_truestrengthindex.IsReady)
        {
            // The current value of _truestrengthindex is represented by itself (_truestrengthindex)
            // or _truestrengthindex.Current.Value
            Plot("TrueStrengthIndex", "truestrengthindex", _truestrengthindex);
            // Plot all properties of abands
            Plot("TrueStrengthIndex", "signal", _truestrengthindex.Signal);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `TrueStrengthIndex` indicator properties:



Indicator History

To get the historical data of the `TrueStrengthIndex` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class TrueStrengthIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private TrueStrengthIndex _tsi;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _tsi = TSI(_symbol, 25, 13, 7, MovingAverageType.Exponential);

        var indicatorHistory = IndicatorHistory(_tsi, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_tsi, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_tsi, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var signal = indicatorHistory.Select(x => ((dynamic)x).Signal).ToList();
    }
}
```

Supported Indicators

Ultimate Oscillator

Introduction

This indicator computes the Ultimate Oscillator (ULTOSC) The Ultimate Oscillator is calculated as explained here: http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:ultimate_oscillator

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ULTOSC Indicator

To create an automatic indicator for `UltimateOscillator` , call the `ULTOSC` helper method from the `QCAlgorithm` class. The `ULTOSC` method creates a `UltimateOscillator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class UltimateOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private UltimateOscillator _ultosc;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ultosc = ULTOSC(_symbol, 5, 10, 20);
    }

    public override void OnData(Slice data)
    {

        if (_ultosc.IsReady)
        {
            // The current value of _ultosc is represented by itself (_ultosc)
            // or _ultosc.Current.Value
            Plot("UltimateOscillator", "ultosc", _ultosc);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `UltimateOscillator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

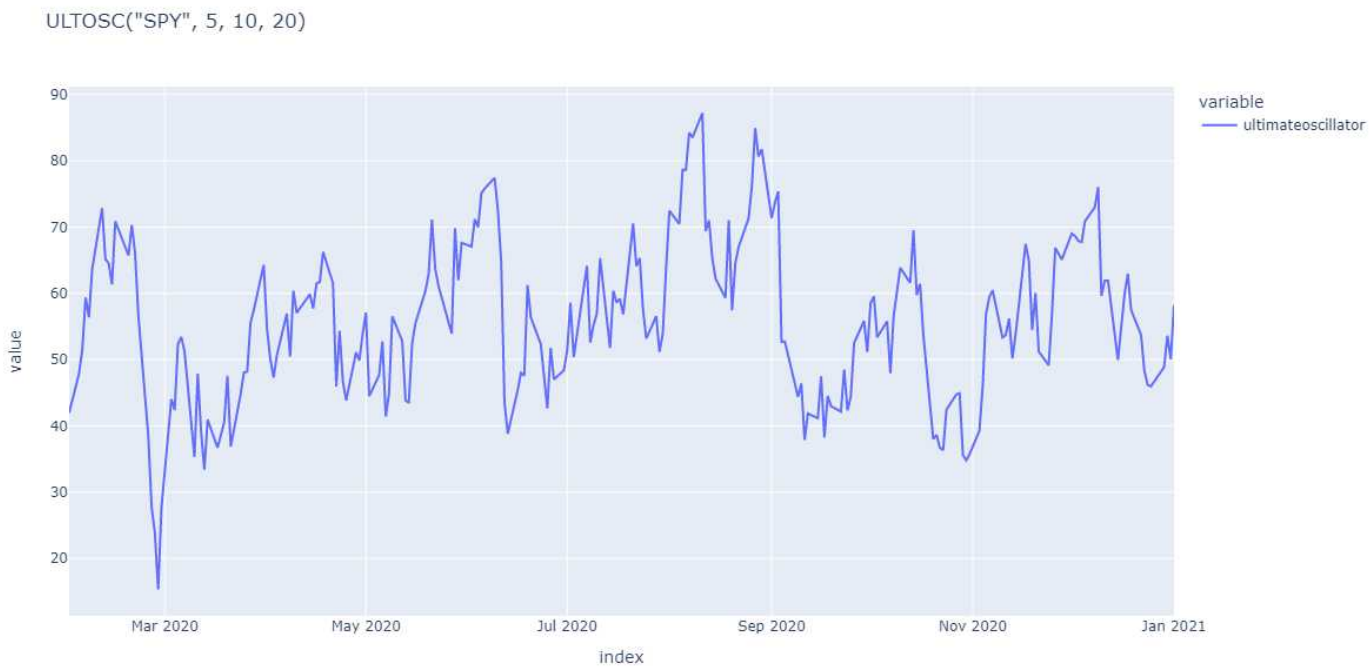
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class UltimateOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private UltimateOscillator _ultimateoscillator;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ultimateoscillator = new UltimateOscillator(5, 10, 20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _ultimateoscillator.Update(bar);

        if (_ultimateoscillator.IsReady)
        {
            // The current value of _ultimateoscillator is represented by itself (_ultimateoscillator)
            // or _ultimateoscillator.Current.Value
            Plot("UltimateOscillator", "ultimateoscillator", _ultimateoscillator);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `UltimateOscillator` indicator properties:



Indicator History

To get the historical data of the `UltimateOscillator` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class UltimateOscillatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private UltimateOscillator _ultosc;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _ultosc = ULTOSC(_symbol, 5, 10, 20);

        var indicatorHistory = IndicatorHistory(_ultosc, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_ultosc, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_ultosc, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Value At Risk

Introduction

This indicator computes 1-day VaR for a specified confidence level and lookback period

To view the implementation of this indicator, see the LEAN GitHub repository .

Using VAR Indicator

To create an automatic indicator for `ValueAtRisk` , call the `VAR` helper method from the `QCAlgorithm` class. The `VAR` method creates a `ValueAtRisk` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class ValueAtRiskAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ValueAtRisk _var;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _var = VAR(_symbol, 252, 0.95);
    }

    public override void OnData(Slice data)
    {

        if (_var.IsReady)
        {
            // The current value of _var is represented by itself (_var)
            // or _var.Current.Value
            Plot("ValueAtRisk", "var", _var);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `ValueAtRisk` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

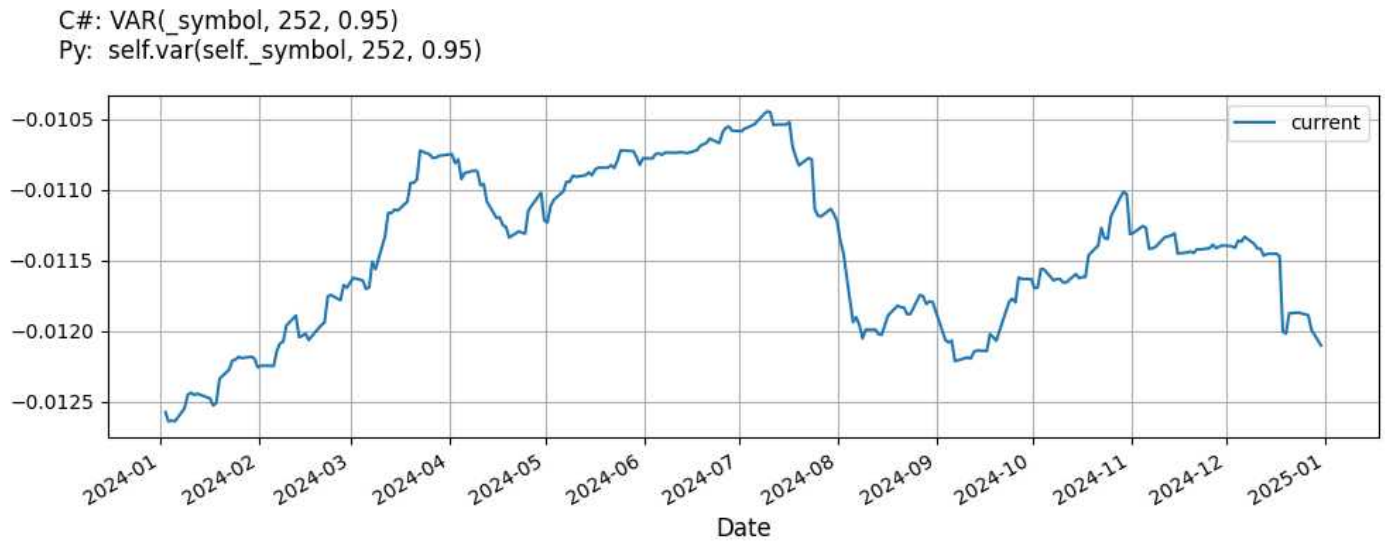
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ValueAtRiskAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ValueAtRisk _valueatrisk;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _valueatrisk = new ValueAtRisk(252, 0.95);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _valueatrisk.Update(bar.EndTime, bar.Close);

        if (_valueatrisk.IsReady)
        {
            // The current value of _valueatrisk is represented by itself (_valueatrisk)
            // or _valueatrisk.Current.Value
            Plot("ValueAtRisk", "valueatrisk", _valueatrisk);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `ValueAtRisk` indicator properties:



Indicator History

To get the historical data of the `ValueAtRisk` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class ValueAtRiskAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ValueAtRisk _var;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _var = VAR(_symbol, 252, 0.95);

        var indicatorHistory = IndicatorHistory(_var, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_var, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_var, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Variable Index Dynamic Average

Introduction

This indicator computes the n-period adaptive weighted moving average indicator. VIDYAi = Pricei x F x ABS(CMOi) + VIDYAi-1 x (1 - F x ABS(CMOi)) where: VIDYAi - is the value of the current period. Pricei - is the source price of the period being calculated. F = 2/(Period_EMA+1) - is a smoothing factor. ABS(CMOi) - is the absolute current value of CMO. VIDYAi-1 - is the value of the period immediately preceding the period being calculated.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using VIDYA Indicator

To create an automatic indicator for `VariableIndexDynamicAverage` , call the `VIDYA` helper method from the `QCAlgorithm` class. The `VIDYA` method creates a `VariableIndexDynamicAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class VariableIndexDynamicAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private VariableIndexDynamicAverage _vidya;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _vidya = VIDYA(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_vidya.IsReady)
        {
            // The current value of _vidya is represented by itself (_vidya)
            // or _vidya.Current.Value
            Plot("VariableIndexDynamicAverage", "vidya", _vidya);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `VariableIndexDynamicAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

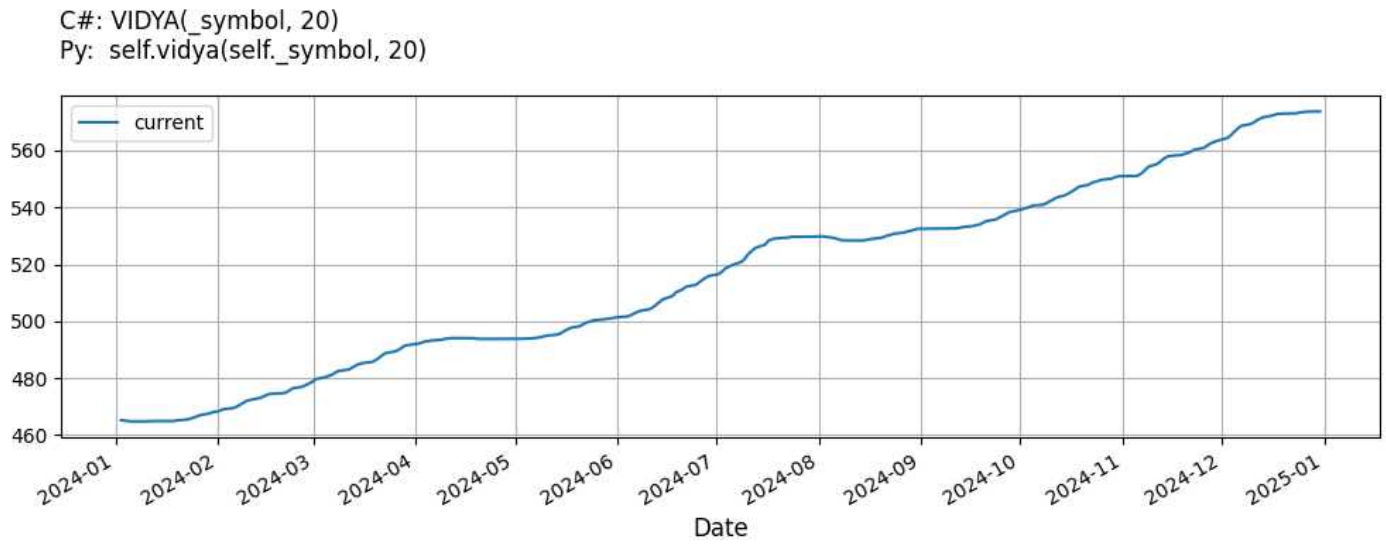
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class VariableIndexDynamicAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private VariableIndexDynamicAverage _variableindexdynamicaverage;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _variableindexdynamicaverage = new VariableIndexDynamicAverage(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _variableindexdynamicaverage.Update(bar.EndTime, bar.Close);

        if (_variableindexdynamicaverage.IsReady)
        {
            // The current value of _variableindexdynamicaverage is represented by itself
(_variableindexdynamicaverage)
            // or _variableindexdynamicaverage.Current.Value
            Plot("VariableIndexDynamicAverage", "variableindexdynamicaverage", _variableindexdynamicaverage);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `VariableIndexDynamicAverage` indicator properties:



Indicator History

To get the historical data of the `VariableIndexDynamicAverage` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class VariableIndexDynamicAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private VariableIndexDynamicAverage _vidya;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _vidya = VIDYA(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_vidya, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_vidya, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_vidya, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Variance

Introduction

This indicator computes the n-period population variance.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using V Indicator

To create an automatic indicator for `Variance` , call the `V` helper method from the `QCAlgorithm` class. The `V` method creates a `Variance` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class VarianceAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Variance _v;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _v = V(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_v.IsReady)
        {
            // The current value of _v is represented by itself (_v)
            // or _v.Current.Value
            Plot("Variance", "v", _v);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `Variance` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

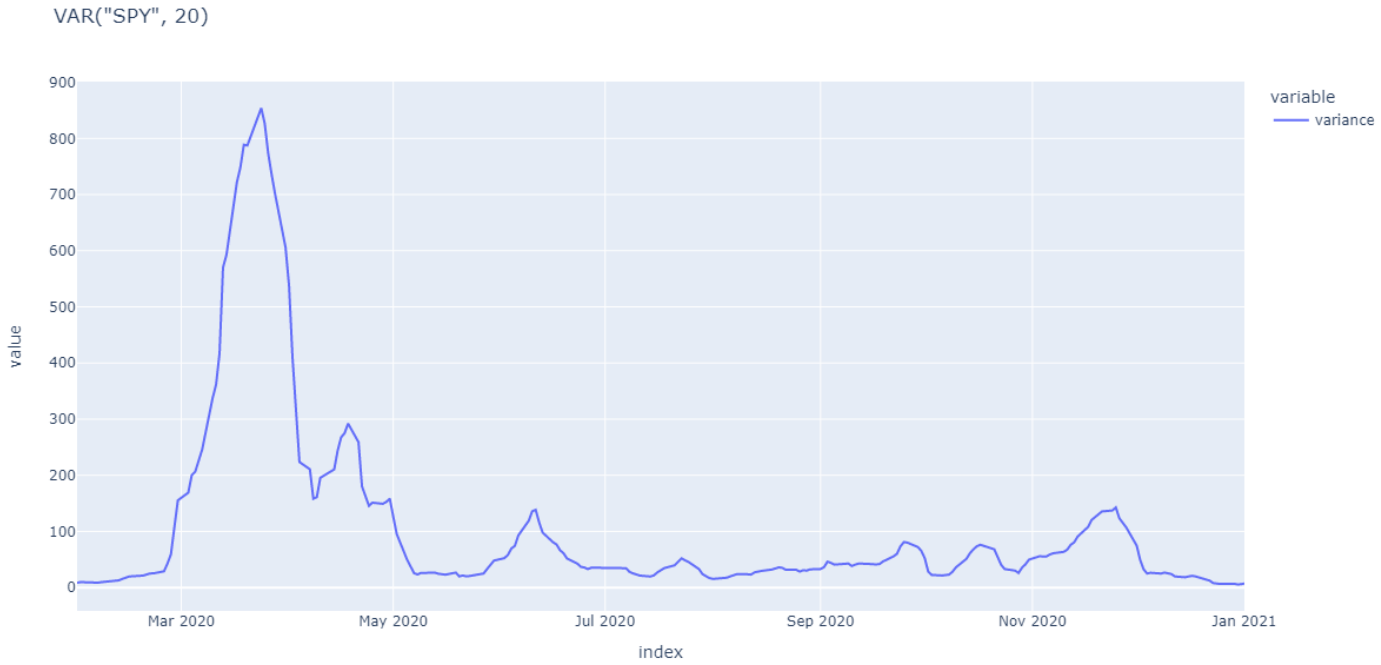
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class VarianceAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Variance _variance;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _variance = new Variance(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _variance.Update(bar.EndTime, bar.Close);

        if (_variance.IsReady)
        {
            // The current value of _variance is represented by itself (_variance)
            // or _variance.Current.Value
            Plot("Variance", "variance", _variance);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `Variance` indicator properties:



Indicator History

To get the historical data of the `Variance` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class VarianceAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Variance _v;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _v = V(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_v, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_v, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_v, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Vega

Introduction

Option Vega indicator that calculate the Vega of an option

To view the implementation of this indicator, see the LEAN GitHub repository .

Using V Indicator

To create an automatic indicator for `Vega` , call the `V` helper method from the `QCAlgorithm` class. The `V` method creates a `Vega` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class VegaAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _option, _mirrorOption;
    private Vega _v;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 20);

        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _option = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920C00564000");
        _mirrorOption = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920P00564000");
        AddOptionContract(_option, Resolution.Daily);
        AddOptionContract(_mirrorOption, Resolution.Daily);

        _v = V(_option, _mirrorOption);
    }

    public override void OnData(Slice data)
    {
        if (_v.IsReady)
        {
            // The current value of _v is represented by itself (_v)
            // or _v.Current.Value
            Plot("Vega", "v", _v);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `Vega` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

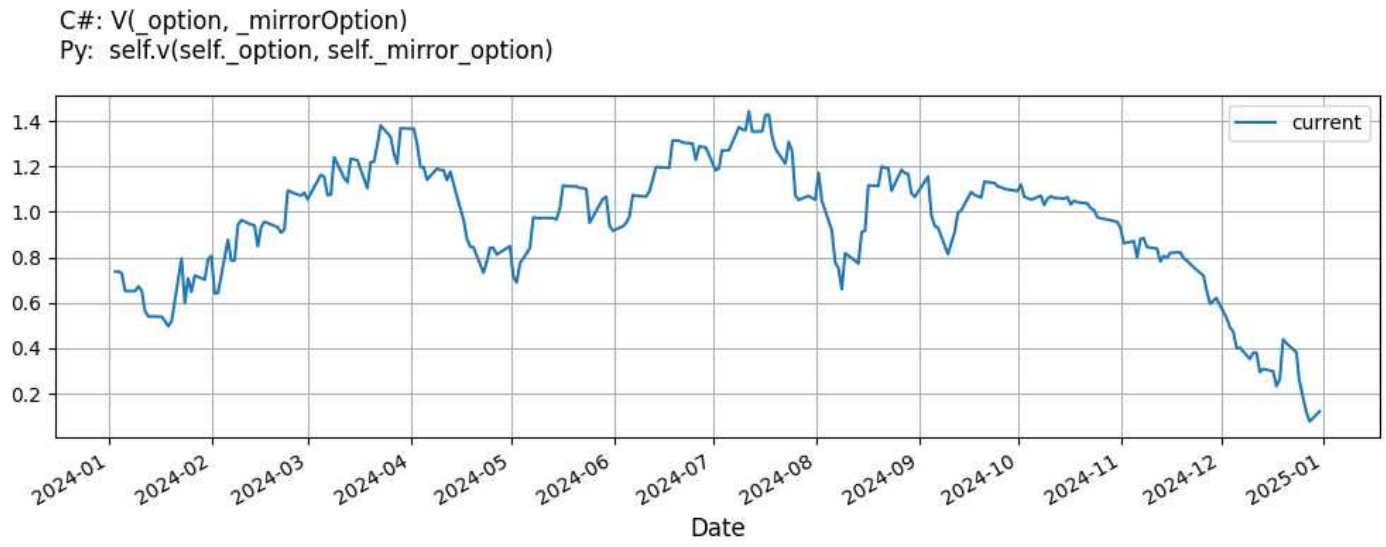
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class VegaAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _option, _mirrorOption;
    private Vega _vega;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 20);

        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        var interestRateModel = new InterestRateProvider();
        var dividendYieldModel = new DividendYieldProvider(_symbol);

        _option = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920C00564000");
        _mirrorOption = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920P00564000");
        AddOptionContract(_option, Resolution.Daily);
        AddOptionContract(_mirrorOption, Resolution.Daily);

        _vega = new Vega(_option, interestRateModel, dividendYieldModel, _mirrorOption);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _vega.Update(new IndicatorDataPoint(_symbol, bar.EndTime, bar.Close));
        if (data.QuoteBars.TryGetValue(_option, out var quoteBar))
            _vega.Update(new IndicatorDataPoint(_option, quoteBar.EndTime, quoteBar.Close));
        if (data.QuoteBars.TryGetValue(_mirrorOption, out quoteBar))
            _vega.Update(new IndicatorDataPoint(_mirrorOption, quoteBar.EndTime, quoteBar.Close));

        if (_vega.IsReady)
        {
            // The current value of _vega is represented by itself (_vega)
            // or _vega.Current.Value
            Plot("Vega", "vega", _vega);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `Vega` indicator properties:



Indicator History

To get the historical data of the `Vega` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory`

method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```C#
public class VegaAlgorithm : QCAlgorithm
{
    private Symbol _symbol, _option, _mirrorOption;
    private Vega _v;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 20);

        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _option = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920C00564000");
        _mirrorOption = SymbolRepresentation.ParseOptionTickerOSI("SPY 240920P00564000");
        AddOptionContract(_option, Resolution.Daily);
        AddOptionContract(_mirrorOption, Resolution.Daily);

        _v = V(_option, _mirrorOption);

        var indicatorHistory = IndicatorHistory(_v, new[] { _symbol, _option, _mirrorOption }, 100,
Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_v, new[] { _symbol, _option, _mirrorOption },
TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_v, new[] { _symbol, _option, _mirrorOption }, new
DateTime(2024, 7, 1), new DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Volume Profile

Introduction

This indicator represents an Indicator of the Market Profile with Volume Profile mode and its attributes

To view the implementation of this indicator, see the LEAN GitHub repository .

Using VP Indicator

To create an automatic indicator for `VolumeProfile` , call the `VP` helper method from the `QCAlgorithm` class. The `VP` method creates a `VolumeProfile` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class VolumeProfileAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private VolumeProfile _vp;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _vp = VP(_symbol, 3, 0.70m, 0.05m);
    }

    public override void OnData(Slice data)
    {

        if (_vp.IsReady)
        {
            // The current value of _vp is represented by itself (_vp)
            // or _vp.Current.Value
            Plot("VolumeProfile", "vp", _vp);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `VolumeProfile` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class VolumeProfileAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private VolumeProfile _volumeprofile;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _volumeprofile = new VolumeProfile(3, 0.70m, 0.05m);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _volumeprofile.Update(bar);

        if (_volumeprofile.IsReady)
        {
            // The current value of _volumeprofile is represented by itself (_volumeprofile)
            // or _volumeprofile.Current.Value
            Plot("VolumeProfile", "volumeprofile", _volumeprofile);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `VolumeProfile` indicator properties:

Indicator History

To get the historical data of the `VolumeProfile` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class VolumeProfileAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private VolumeProfile _vp;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _vp = VP(_symbol, 3, 0.70m, 0.05m);

        var indicatorHistory = IndicatorHistory(_vp, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_vp, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_vp, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Volume Weighted Average Price Indicator

Introduction

Volume Weighted Average Price (VWAP) Indicator: It is calculated by adding up the dollars traded for every transaction (price multiplied by number of shares traded) and then dividing by the total shares traded for the day.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using VWAP Indicator

To create an automatic indicator for `VolumeWeightedAveragePriceIndicator` , call the `VWAP` helper method from the `QCAlgorithm` class. The `VWAP` method creates a `VolumeWeightedAveragePriceIndicator` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class VolumeWeightedAveragePriceIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private VolumeWeightedAveragePriceIndicator _vwap;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _vwap = VWAP(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_vwap.IsReady)
        {
            // The current value of _vwap is represented by itself (_vwap)
            // or _vwap.Current.Value
            Plot("VolumeWeightedAveragePriceIndicator", "vwap", _vwap);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `VolumeWeightedAveragePriceIndicator` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

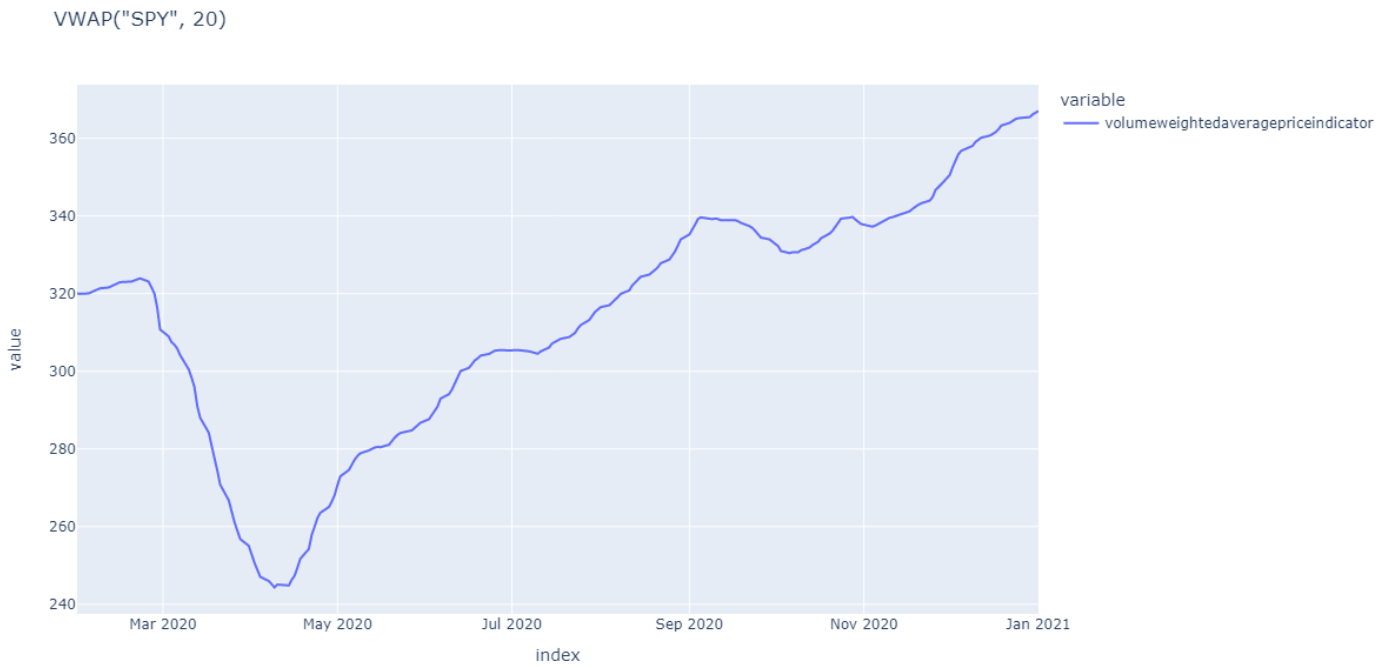
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class VolumeWeightedAveragePriceIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private VolumeWeightedAveragePriceIndicator _volumeweightedaveragepriceindicator;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _volumeweightedaveragepriceindicator = new VolumeWeightedAveragePriceIndicator(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _volumeweightedaveragepriceindicator.Update(bar);

        if (_volumeweightedaveragepriceindicator.IsReady)
        {
            // The current value of _volumeweightedaveragepriceindicator is represented by itself
(_volumeweightedaveragepriceindicator)
            // or _volumeweightedaveragepriceindicator.Current.Value
            Plot("VolumeWeightedAveragePriceIndicator", "volumeweightedaveragepriceindicator",
_volumeweightedaveragepriceindicator);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `VolumeWeightedAveragePriceIndicator` indicator properties:



Indicator History

To get the historical data of the `VolumeWeightedAveragePriceIndicator` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class VolumeWeightedAveragePriceIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private VolumeWeightedAveragePriceIndicator _vwap;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _vwap = VWAP(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_vwap, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_vwap, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_vwap, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Volume Weighted Moving Average

Introduction

This indicator computes the Volume Weighted Moving Average (VWMA) It is a technical analysis indicator used by traders to determine the average price of an asset over a given period of time, taking into account both price and volume.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using VWMA Indicator

To create an automatic indicator for `VolumeWeightedMovingAverage` , call the `VWMA` helper method from the `QCAlgorithm` class. The `VWMA` method creates a `VolumeWeightedMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class VolumeWeightedMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private VolumeWeightedMovingAverage _vwma;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _vwma = VWMA(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_vwma.IsReady)
        {
            // The current value of _vwma is represented by itself (_vwma)
            // or _vwma.Current.Value
            Plot("VolumeWeightedMovingAverage", "vwma", _vwma);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `VolumeWeightedMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

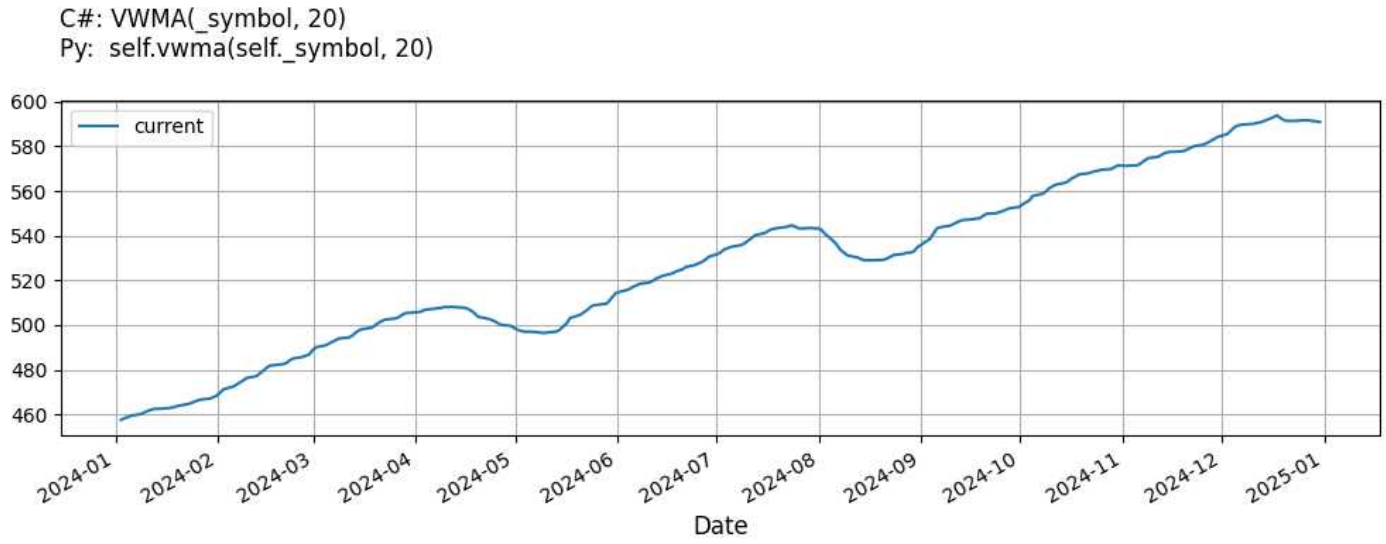
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class VolumeWeightedMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private VolumeWeightedMovingAverage _volumeweightedmovingaverage;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _volumeweightedmovingaverage = new VolumeWeightedMovingAverage(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _volumeweightedmovingaverage.Update(bar);

        if (_volumeweightedmovingaverage.IsReady)
        {
            // The current value of _volumeweightedmovingaverage is represented by itself
(_volumeweightedmovingaverage)
            // or _volumeweightedmovingaverage.Current.Value
            Plot("VolumeWeightedMovingAverage", "volumeweightedmovingaverage", _volumeweightedmovingaverage);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `VolumeWeightedMovingAverage` indicator properties:



Indicator History

To get the historical data of the `VolumeWeightedMovingAverage` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class VolumeWeightedMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private VolumeWeightedMovingAverage _vwma;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _vwma = VWMA(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_vwma, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_vwma, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_vwma, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Vortex

Introduction

This indicator represents the Vortex Indicator, which identifies the start and continuation of market trends. It includes components that capture positive (upward) and negative (downward) trend movements. This indicator compares the ranges within the current period to previous periods to calculate upward and downward movement trends.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using VTX Indicator

To create an automatic indicator for `Vortex` , call the `VTX` helper method from the `QCAlgorithm` class. The `VTX` method creates a `Vortex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

C#

```
public class VortexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Vortex _vtx;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _vtx = VTX(_symbol, 14);
    }

    public override void OnData(Slice data)
    {

        if (_vtx.IsReady)
        {
            // The current value of _vtx is represented by itself (_vtx)
            // or _vtx.Current.Value
            Plot("Vortex", "vtx", _vtx);
            // Plot all properties of abands
            Plot("Vortex", "plusvortex", _vtx.PlusVortex);
            Plot("Vortex", "minusvortex", _vtx.MinusVortex);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `Vortex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

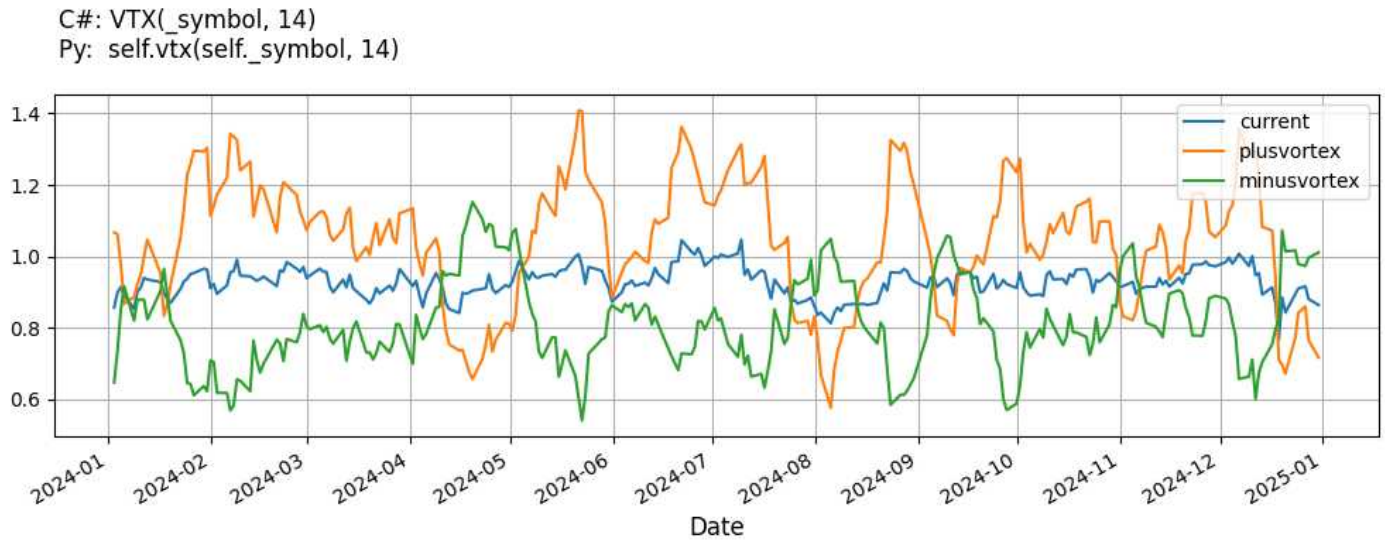
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class VortexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Vortex _vortex;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _vortex = new Vortex(14);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _vortex.Update(bar);

        if (_vortex.IsReady)
        {
            // The current value of _vortex is represented by itself (_vortex)
            // or _vortex.Current.Value
            Plot("Vortex", "vortex", _vortex);
            // Plot all properties of abands
            Plot("Vortex", "plusvortex", _vortex.PlusVortex);
            Plot("Vortex", "minusvortex", _vortex.MinusVortex);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `Vortex` indicator properties:



Indicator History

To get the historical data of the `Vortex` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class VortexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private Vortex _vtx;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _vtx = VTX(_symbol, 14);

        var indicatorHistory = IndicatorHistory(_vtx, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_vtx, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_vtx, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var plusVortex = indicatorHistory.Select(x => ((dynamic)x).PlusVortex).ToList();
        var minusVortex = indicatorHistory.Select(x => ((dynamic)x).MinusVortex).ToList();
    }
}
```

Supported Indicators

Wilder Accumulative Swing Index

Introduction

This indicator calculates the Accumulative Swing Index (ASI) as defined by Welles Wilder in his book 'New Concepts in Technical Trading Systems'. $ASI_t = ASI_{t-1} + SI_t$ Where: ASI_{t-1} The for the previous period. SI_t The calculated for the current period.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ASI Indicator

To create an automatic indicator for `WilderAccumulativeSwingIndex` , call the `ASI` helper method from the `QCAlgorithm` class. The `ASI` method creates a `WilderAccumulativeSwingIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```C#
public class WilderAccumulativeSwingIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private WilderAccumulativeSwingIndex _asi;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _asi = ASI(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_asi.IsReady)
        {
            // The current value of _asi is represented by itself (_asi)
            // or _asi.Current.Value
            Plot("WilderAccumulativeSwingIndex", "asi", _asi);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `WilderAccumulativeSwingIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

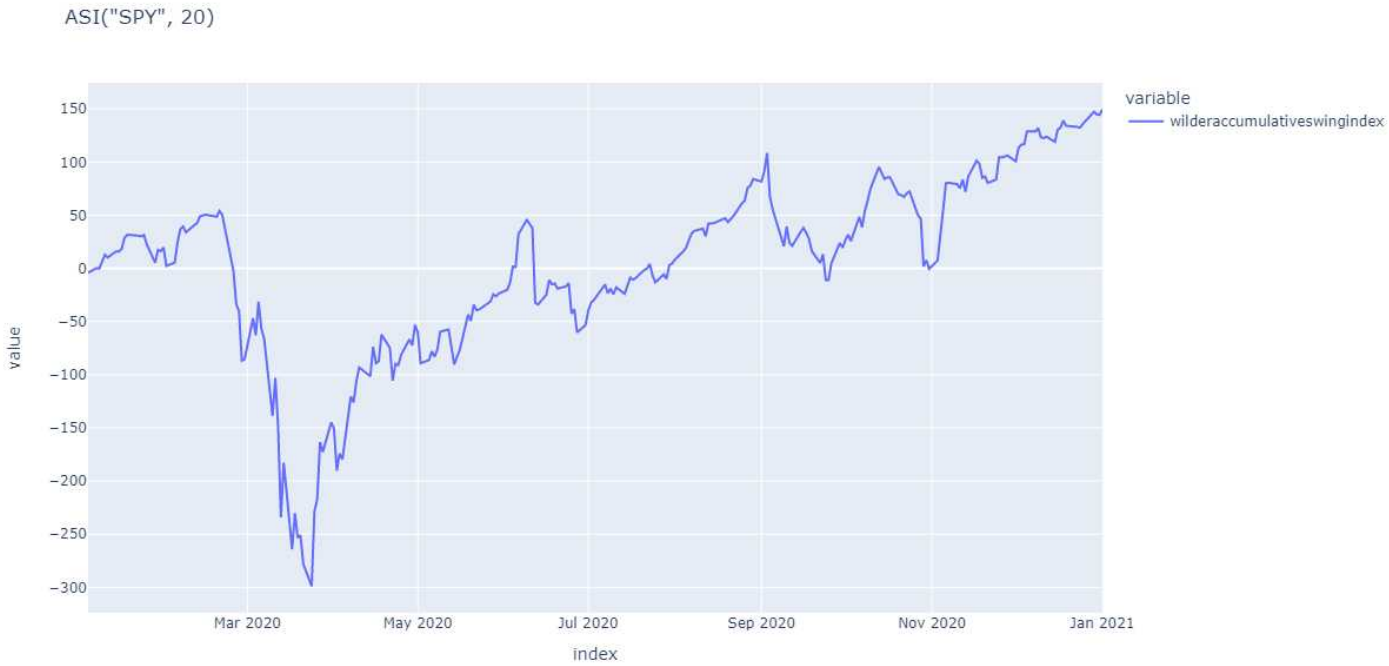
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class WilderAccumulativeSwingIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private WilderAccumulativeSwingIndex _wilderaccumulativeswingindex;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _wilderaccumulativeswingindex = new WilderAccumulativeSwingIndex(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _wilderaccumulativeswingindex.Update(bar);

        if (_wilderaccumulativeswingindex.IsReady)
        {
            // The current value of _wilderaccumulativeswingindex is represented by itself
(_wilderaccumulativeswingindex)
            // or _wilderaccumulativeswingindex.Current.Value
            Plot("WilderAccumulativeSwingIndex", "wilderaccumulativeswingindex", _wilderaccumulativeswingindex);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `WilderAccumulativeSwingIndex` indicator properties:



Indicator History

To get the historical data of the `WilderAccumulativeSwingIndex` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class WilderAccumulativeSwingIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private WilderAccumulativeSwingIndex _asi;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _asi = ASI(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_asi, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_asi, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_asi, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Wilder Moving Average

Introduction

This indicator represents the moving average indicator defined by Welles Wilder in his book: New Concepts in Technical Trading Systems.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using WWMA Indicator

To create an automatic indicator for `WilderMovingAverage` , call the `WWMA` helper method from the `QCAlgorithm` class. The `WWMA` method creates a `WilderMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class WilderMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private WilderMovingAverage _wwma;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _wwma = WWMA(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_wwma.IsReady)
        {
            // The current value of _wwma is represented by itself (_wwma)
            // or _wwma.Current.Value
            Plot("WilderMovingAverage", "wwma", _wwma);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `WilderMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

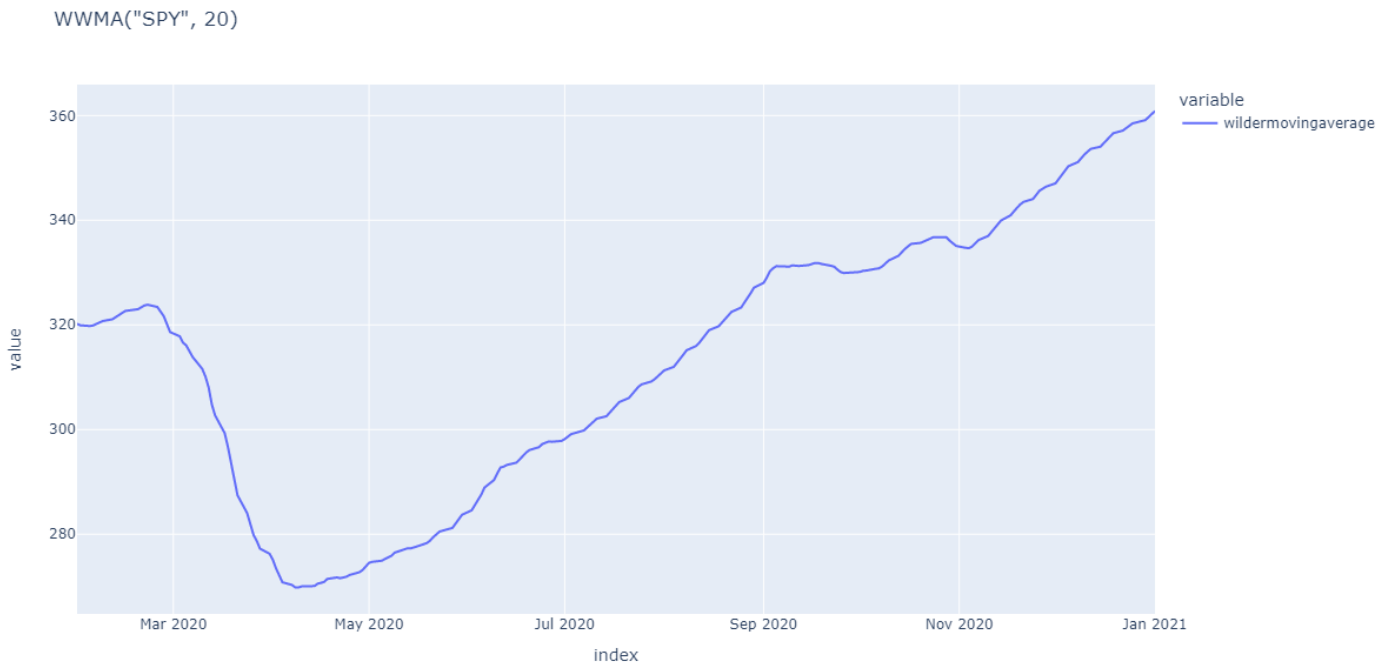
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class WilderMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private WilderMovingAverage _wildermovingaverage;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _wildermovingaverage = new WilderMovingAverage(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _wildermovingaverage.Update(bar.EndTime, bar.Close);

        if (_wildermovingaverage.IsReady)
        {
            // The current value of _wildermovingaverage is represented by itself (_wildermovingaverage)
            // or _wildermovingaverage.Current.Value
            Plot("WilderMovingAverage", "wildermovingaverage", _wildermovingaverage);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `WilderMovingAverage` indicator properties:



Indicator History

To get the historical data of the `WilderMovingAverage` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class WilderMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private WilderMovingAverage _wwma;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _wwma = WWMA(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_wwma, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_wwma, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_wwma, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Wilder Swing Index

Introduction

This indicator calculates the Swing Index (SI) as defined by Welles Wilder in his book 'New Concepts in Technical Trading Systems'. SI_t = 50 * (N / R) * (K / T) Where: N Equals: C_t - C_{t-1} + 0.5 * (C_t - O_t) + 0.25 * (C_{t-1} - O_{t-1}) *See* R Found by selecting the expression with the largest value and then using the corresponding formula. Expression => Formula $|H_t - C_{t-1}|$ => $|H_t - C_t|$ - 0.5 * $|L_t - C_{t-1}|$ + 0.25 * $|C_{t-1} - O_{t-1}|$ $|L_t - C_{t-1}|$ => $|L_t - C_t|$ - 0.5 * $|H_t - C_{t-1}|$ + 0.25 * $|C_{t-1} - O_{t-1}|$ $|H_t - L_t|$ => $|H_t - L_{t-1}|$ + 0.25 * $|C_{t-1} - O_{t-1}|$ *See* K Found by selecting the larger of the two expressions: $|H_t - C_{t-1}|$, $|L_t - C_{t-1}|$ *See* T The limit move, or the maximum change in price during the time period for the bar. Passed as limitMove via the constructor. *See*

To view the implementation of this indicator, see the LEAN GitHub repository .

Using SI Indicator

To create an automatic indicator for `WilderSwingIndex` , call the `SI` helper method from the `QCAlgorithm` class. The `SI` method creates a `WilderSwingIndex` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class WilderSwingIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private WilderSwingIndex _si;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _si = SI(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_si.IsReady)
        {
            // The current value of _si is represented by itself (_si)
            // or _si.Current.Value
            Plot("WilderSwingIndex", "si", _si);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `WilderSwingIndex` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

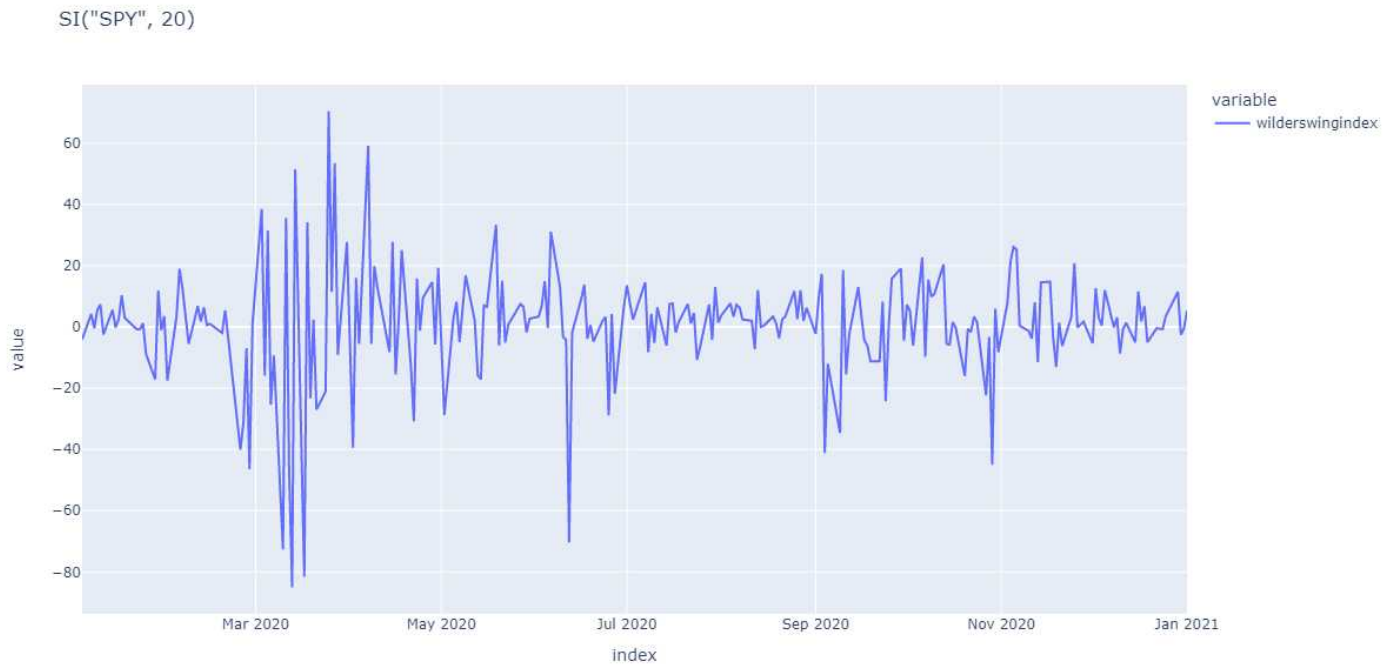
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class WilderSwingIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private WilderSwingIndex _wilderswingindex;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _wilderswingindex = new WilderSwingIndex(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _wilderswingindex.Update(bar);

        if (_wilderswingindex.IsReady)
        {
            // The current value of _wilderswingindex is represented by itself (_wilderswingindex)
            // or _wilderswingindex.Current.Value
            Plot("WilderSwingIndex", "wilderswingindex", _wilderswingindex);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `WilderSwingIndex` indicator properties:



Indicator History

To get the historical data of the `WilderSwingIndex` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class WilderSwingIndexAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private WilderSwingIndex _si;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _si = SI(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_si, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_si, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_si, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Williams Percent R

Introduction

Williams %R, or just %R, is the current closing price in relation to the high and low of the past N days (for a given N). The value of this indicator fluctuates between -100 and 0. The symbol is said to be oversold when the oscillator is below -80%, and overbought when the oscillator is above -20%.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using WILR Indicator

To create an automatic indicator for `WilliamsPercentR` , call the `WILR` helper method from the `QCAlgorithm` class. The `WILR` method creates a `WilliamsPercentR` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class WilliamsPercentRAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private WilliamsPercentR _wilr;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _wilr = WILR(_symbol, 20);
    }

    public override void OnData(Slice data)
    {

        if (_wilr.IsReady)
        {
            // The current value of _wilr is represented by itself (_wilr)
            // or _wilr.Current.Value
            Plot("WilliamsPercentR", "wilr", _wilr);
            // Plot all properties of abands
            Plot("WilliamsPercentR", "maximum", _wilr.Maximum);
            Plot("WilliamsPercentR", "minimum", _wilr.Minimum);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `WilliamsPercentR` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

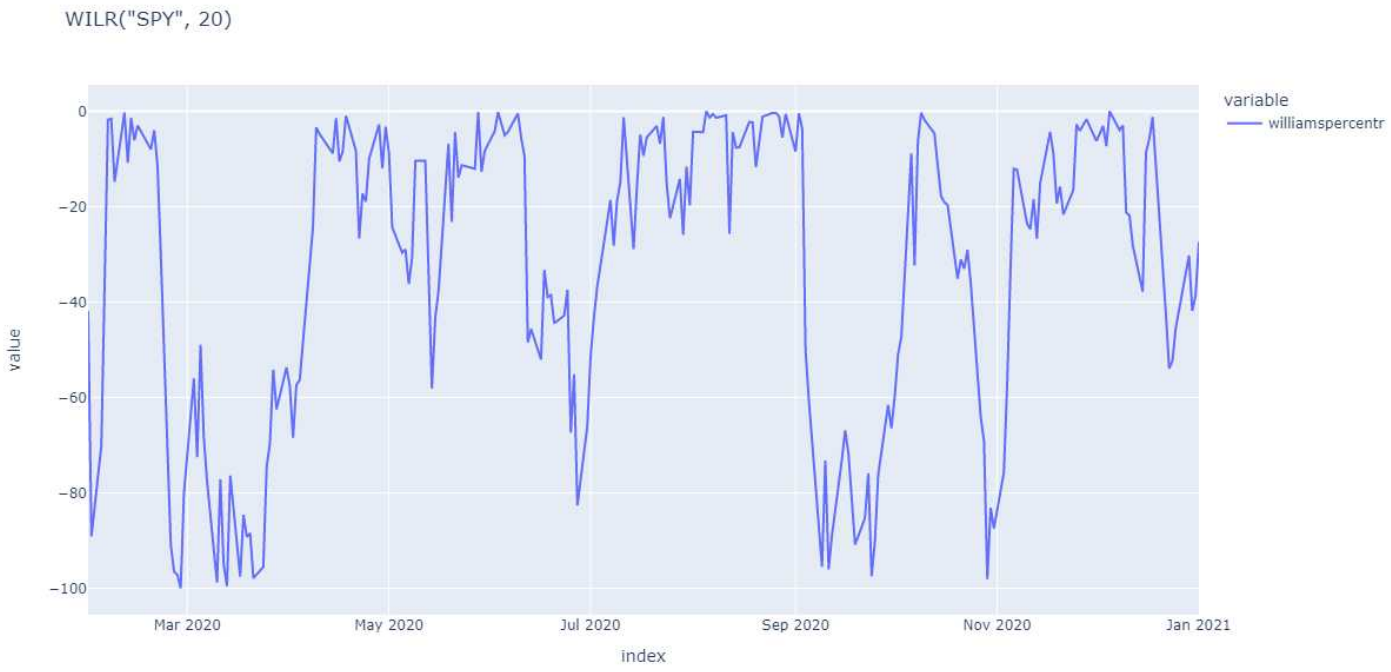
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```
C#
public class WilliamsPercentRAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private WilliamsPercentR _williamspercentr;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _williamspercentr = new WilliamsPercentR(20);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _williamspercentr.Update(bar);

        if (_williamspercentr.IsReady)
        {
            // The current value of _williamspercentr is represented by itself (_williamspercentr)
            // or _williamspercentr.Current.Value
            Plot("WilliamsPercentR", "williamspercentr", _williamspercentr);
            // Plot all properties of abands
            Plot("WilliamsPercentR", "maximum", _williamspercentr.Maximum);
            Plot("WilliamsPercentR", "minimum", _williamspercentr.Minimum);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `WilliamsPercentR` indicator properties:



Indicator History

To get the historical data of the `WilliamsPercentR` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class WilliamsPercentRAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private WilliamsPercentR _wilr;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _wilr = WILR(_symbol, 20);

        var indicatorHistory = IndicatorHistory(_wilr, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_wilr, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_wilr, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);

        // Access all attributes of indicatorHistory
        var maximum = indicatorHistory.Select(x => ((dynamic)x).Maximum).ToList();
        var minimum = indicatorHistory.Select(x => ((dynamic)x).Minimum).ToList();
    }
}
```

Supported Indicators

Zero Lag Exponential Moving Average

Introduction

This indicator represents the zero lag moving average indicator (ZLEMA) ie a technical indicator that aims is to eliminate the inherent lag associated to all trend following indicators which average a price over time.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ZLEMA Indicator

To create an automatic indicator for `ZeroLagExponentialMovingAverage` , call the `ZLEMA` helper method from the `QCAlgorithm` class. The `ZLEMA` method creates a `ZeroLagExponentialMovingAverage` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class ZeroLagExponentialMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ZeroLagExponentialMovingAverage _zlema;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _zlema = ZLEMA(_symbol, 10);
    }

    public override void OnData(Slice data)
    {

        if (_zlema.IsReady)
        {
            // The current value of _zlema is represented by itself (_zlema)
            // or _zlema.Current.Value
            Plot("ZeroLagExponentialMovingAverage", "zlema", _zlema);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `ZeroLagExponentialMovingAverage` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

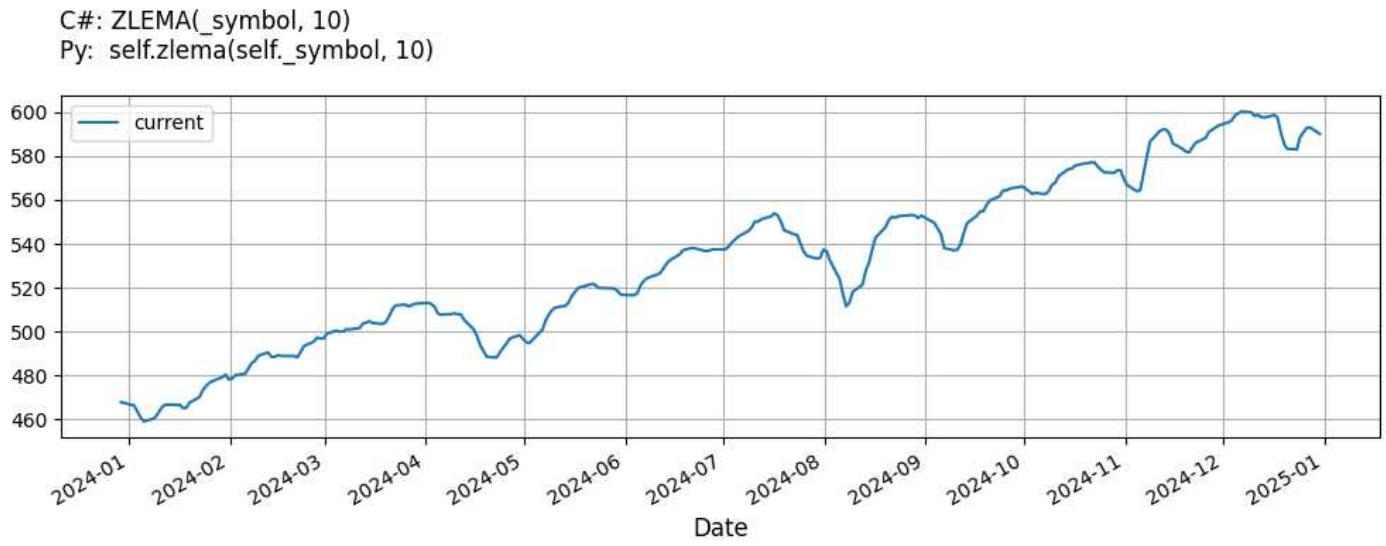
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ZeroLagExponentialMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ZeroLagExponentialMovingAverage _zerolagexponentialmovingaverage;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _zerolagexponentialmovingaverage = new ZeroLagExponentialMovingAverage(10);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _zerolagexponentialmovingaverage.Update(bar.EndTime, bar.Close);

        if (_zerolagexponentialmovingaverage.IsReady)
        {
            // The current value of _zerolagexponentialmovingaverage is represented by itself
(_zerolagexponentialmovingaverage)
            // or _zerolagexponentialmovingaverage.Current.Value
            Plot("ZeroLagExponentialMovingAverage", "zerolagexponentialmovingaverage",
_zerolagexponentialmovingaverage);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `ZeroLagExponentialMovingAverage` indicator properties:



Indicator History

To get the historical data of the `ZeroLagExponentialMovingAverage` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class ZeroLagExponentialMovingAverageAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ZeroLagExponentialMovingAverage _zlema;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _zlema = ZLEMA(_symbol, 10);

        var indicatorHistory = IndicatorHistory(_zlema, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_zlema, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_zlema, _symbol, new DateTime(2024, 7, 1), new
DateTime(2024, 7, 5), Resolution.Minute);
    }
}
```

Supported Indicators

Zig Zag

Introduction

The ZigZag indicator identifies significant turning points in price movements, filtering out noise using a sensitivity threshold and a minimum trend length. It alternates between high and low pivots to determine market trends.

To view the implementation of this indicator, see the LEAN GitHub repository .

Using ZZ Indicator

To create an automatic indicator for `ZigZag` , call the `ZZ` helper method from the `QCAlgorithm` class. The `ZZ` method creates a `ZigZag` object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. In most cases, you should call the helper method in the `Initialize` method.

```
C#
public class ZigZagAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ZigZag _zz;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _zz = ZZ(_symbol, 0.05m, 1);
    }

    public override void OnData(Slice data)
    {

        if (_zz.IsReady)
        {
            // The current value of _zz is represented by itself (_zz)
            // or _zz.Current.Value
            Plot("ZigZag", "zz", _zz);
        }
    }
}
```

For more information about this method, see the QCAlgorithm class .

You can manually create a `ZigZag` indicator, so it doesn't automatically update. Manual indicators let you update their values with any data you choose.

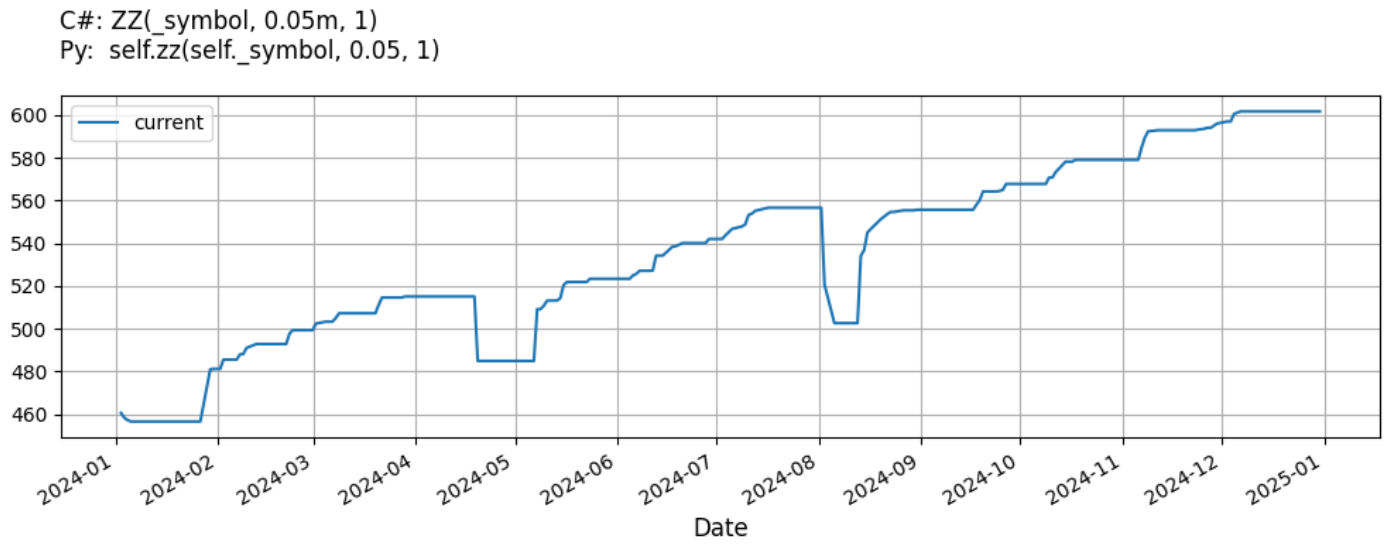
Updating your indicator manually enables you to control when the indicator is updated and what data you use to update it. To manually update the indicator, call the `Update` method. The indicator will only be ready after you prime it with enough data.

```C#
public class ZigZagAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ZigZag _zigzag;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _zigzag = new ZigZag(0.05m, 1);
    }

    public override void OnData(Slice data)
    {
        if (data.Bars.TryGetValue(_symbol, out var bar))
            _zigzag.Update(bar);

        if (_zigzag.IsReady)
        {
            // The current value of _zigzag is represented by itself (_zigzag)
            // or _zigzag.Current.Value
            Plot("ZigZag", "zigzag", _zigzag);
        }
    }
}
```

For more information about this indicator, see its reference .

Visualization

The following plot shows values for some of the `ZigZag` indicator properties:



Indicator History

To get the historical data of the `ZigZag` indicator, call the `IndicatorHistory` method. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

C#

```
public class ZigZagAlgorithm : QCAlgorithm
{
    private Symbol _symbol;
    private ZigZag _zz;

    public override void Initialize()
    {
        _symbol = AddEquity("SPY", Resolution.Daily).Symbol;
        _zz = ZZ(_symbol, 0.05m, 1);

        var indicatorHistory = IndicatorHistory(_zz, _symbol, 100, Resolution.Minute);
        var timeSpanIndicatorHistory = IndicatorHistory(_zz, _symbol, TimeSpan.FromDays(10), Resolution.Minute);
        var timePeriodIndicatorHistory = IndicatorHistory(_zz, _symbol, new DateTime(2024, 7, 1), new DateTime(2024,
7, 5), Resolution.Minute);
    }
}
```

Indicators

Key Concepts

Introduction

Indicators translate a stream of data points into a numerical value you can use to detect trading opportunities. LEAN provides more than 100 pre-built technical indicators and candlestick patterns you can use in your algorithms. To view a list of all the indicators, see the Supported Indicators . One simple indicator to learn is the Identity indicator, which just returns the value of the asset.

```
var pep = Identity("PEP"); // Pepsi ticker
```

C#

Design Philosophy

Technical indicators take in a stream of data points, `Bar` objects, or `TradeBar` objects and produce a continuous value. Candlestick patterns take in a stream of `Bar` or `TradeBar` objects and produce a value that's either 0, 1, or -1 to signify the presence of the pattern.

You can configure technical indicators and candlestick patterns to receive manual or automatic updates. With manual indicators, you update the indicator with whatever data you want and whenever you want. With automatic indicators, the algorithm automatically uses the security data to update the indicator value.

LEAN's indicators are implemented "on-line". This means as data pipes through, it updates the internal state of the indicator. Other platforms perform indicator math on an array of values in bulk which can be very inefficient when working on large volumes of data. You need to warm up your indicators and prime them with data to get them ready.

Indicator Types

You can classify an indicator as either a data point, bar, or TradeBar indicator. Their classification depends on the type of data they receive.

Data Point Indicators

Data point indicators use `IndicatorDataPoint` objects to compute their value. Some examples of data point indicators include the following:

- SimpleMovingAverage
- ExponentialMovingAverage
- Momentum

Bar Indicators

Bar indicators use `QuoteBar` or `TradeBar` objects to compute their value. Since Forex and CFD securities don't have `TradeBar` data, they use bar indicators. Candlestick patterns are examples of bar indicators. Some other examples of bar indicators include the following:

- WilliamsPercentR
- Stochastics

- AverageTrueRange

TradeBar Indicators

`TradeBar` indicators use `TradeBar` objects to compute their value. Some `TradeBar` indicators use the volume property of the `TradeBar` to compute their value. Some examples of `TradeBar` indicators include the following:

- Beta
- AdvanceDeclineVolumeRatio
- VolumeWeightedAveragePriceIndicator

Naming Convention

The class name to create a manual indicator is the indicator name spelled in pascal case. For example, to create a manual simple moving average indicator , use the `SimpleMovingAverage` class.

The method to create an automatic indicator is usually named after the acronym of the indicator name. For example, to create an automatic simple moving average indicator , use the `SMA` method.

To view the class name and shortcut method of each indicator, see the Supported Indicators .

Create Indicators

There are two ways to create indicators. You can create indicators with the indicator constructor or use a helper method in the `QCAlgorithm` class. If you use the helper method, the indicator automatically updates as your algorithm receives new data.

```
C#
private BollingerBands _manualBB;
private BollingerBands _autoBB;
private Symbol _symbol;
public override void Initialize()
{
    // Create a manual indicator with the indicator constructor
    _manualBB = new BollingerBands(20, 2);
    // Create an automatic indicator with the helper method
    _symbol = AddCrypto("BTCUSD").Symbol;
    _autoBB = BB(_symbol, 20, 2, Resolution.Daily);
}
```

Set Historical Values Window Size

Indicators have a built-in `RollingWindow` that stores their historical values. The default window size is 2. To change the window size, set the `Window.Size` member.

```
C#
// Increase the window size to 10 for the Bollinger Band
_manualBB.Window.Size = 10;
_autoBB.Window.Size = 10;

// Increase the Size to 20 for the middle band
_manualBB.MiddleBand.Window.Size = 20;
_autoBB.MiddleBand.Window.Size = 20;
```

When you decrease the size, it removes the oldest values that no longer fit in the `RollingWindow` . When you explicitly increase the `Size` member, it doesn't automatically add any new elements to the `RollingWindow` . However, if you set the value of an index in the `RollingWindow` and the index doesn't currently exist, it fills the empty indices with `null` . For example, the following code increases the `Size` to 10, sets the 10th element to the indicator current value, and sets the 6th-9th elements to `null` :

```
_manualBB.Window[9] = _manualBB.Current;
_autoBB.Window[9] = _autoBB.Current;
```

Check Readiness

Indicators aren't always ready when you first create them. The length of time it takes to trust the indicator values depends on the indicator period. To know if an indicator is ready to use, use the `IsReady` property.

```
if (!_autoBB.IsReady) return;
```

To get the number of samples the indicator has processed, use the `Samples` property.

```
var samples = _autoBB.Samples;
```

The `RollingWindow` objects that store the historical values may not be full when the indicator is ready to use. To check if the `Window` is full, use its `IsReady` flag.

```
if (!_autoBB.Window.IsReady) return;
```

Track Update Events

When an indicator receives a new data point and updates its state, it triggers an update event. To be notified of update events, attach an event handler to the indicator object. The event handler receives 2 arguments: the indicator object and the latest `IndicatorDataPoint` it produced.

```
// After you create the indicator, attach an event handler
_autoBB.Updated += updateEventHandler;

// Define the event handler within the same class
private void updateEventHandler(object indicator, IndicatorDataPoint indicatorDataPoint)
{
    if (indicator.IsReady)
    {
        Plot("Indicator", "Value", indicatorDataPoint.Value);
    }
}
```

Get Current Values

To access the indicator value, use the `Current.Value` property. Some indicators have one output and some indicators have multiple outputs. The `SimpleMovingAverage` indicator only has one output, the average price over the last n periods, so the `Current.Value` property returns this value. The `BollingerBand` indicator has multiple outputs because it has a simple moving average, an upper band, and a lower band. For indicators that have multiple outputs, refer to the Supported Indicators to see how to access the output values.

```
C#
var sma = _sma.Current.Value;

var currentPrice = _autoBB.Current.Value;
var bbUpperBand = _autoBB.UpperBand.Current.Value;
var bbLowerBand = _autoBB.LowerBand.Current.Value;
```

You can implicitly cast indicators to the decimal version of their `Current.Value` property.

```
C#
if (_sma > _autoBB.UpperBand)
{
    SetHoldings(_symbol, -0.1);
}
```

Get Historical Values

To access historical indicator values, you can use the indicator's built-in `RollingWindow` or you can call the `IndicatorHistory` method to request values over a specific period of time.

Get Trailing Values

To access trailing indicator values, use reverse list access semantics with positive and negative indexing. The current (most recent) indicator value is at index 0, the previous value is at index 1 or the `Previous` property, and so on until the length of the window. By default, the window length is 2. If you index the window with a number that's greater than the maximum index of the window, you get a `null` value and the window length grows to support the new index.

```
C#
var currentBB = _autoBB.Current; // or _autoBB[0];
var previousBB = _autoBB.Previous; // or _autoBB[1];
var oldestBB = _autoBB.Window[-1];
var nullValue = _autoBB.Window[100];

var previousUpperBand = _autoBB.UpperBand.Previous;
var oldestUpperBand = _autoBB.UpperBand.Window[-1];
```

To access all of an indicator's historical values in the window, iterate through the indicator object.

```
C#
foreach (var value in _autoBB)
{
    Log(value.ToString());
}
```

Request Historical Values

The `IndicatorHistory` method also provides historical indicator values. This method resets your indicator, makes a history request , and updates the indicator with the historical data. Just like with regular history requests, the `IndicatorHistory` method supports time periods based on a trailing number of bars, a trailing period of time, or a defined period of time. If you don't provide a `resolution` argument, it defaults to match the resolution of the security subscription.

```
// Get historical indicator values.
var history = IndicatorHistory(_sma, _symbol, TimeSpan.FromDays(30));

// Iterate through the historical values.
foreach (var indicatorDataPoint in history)
{
    var t = indicatorDataPoint.EndTime;
    var value = indicatorDataPoint.Value;
}
```

To make the `IndicatorHistory` method update the indicator with an alternative price field instead of the close (or mid-price) of each bar, pass a `selector` argument.

```
var history = IndicatorHistory(_sma, _symbol, TimeSpan.FromDays(30), selector: Field.Volume);
```

Some indicators require the prices of two assets to compute their value (for example, Beta). In this case, pass a list of the `Symbol` objects to the method.

```
var spy = AddEquity("SPY").Symbol;
var aapl = AddEquity("AAPL").Symbol;
var beta = new Beta("", aapl, spy, 21);
var history = IndicatorHistory(beta, new[] {spy, aapl}, 10, Resolution.Daily);
```

If the indicator is an Option indicator with an underlying Equity, the `IndicatorHistory` uses the `ScaledRaw` data normalization mode when requesting historical prices.

If you already have a list of Slice objects, you can pass them to the `IndicatorHistory` method to avoid the internal history request.

```
var slices = History(new[] {_symbol}, 30, Resolution.Daily);
var history = IndicatorHistory(_sma, slices);
```

Reset Indicators

To reset indicators, call the `Reset` method.

```
_autoBB.Reset();
```

The process to warm up your indicator depends on if it's a manual or automatic indicator .

If you are live trading Equities or backtesting Equities without the adjusted data normalization mode , reset your indicators when splits and dividends occur. If a split or dividend occurs, the data in your indicators becomes invalid because it doesn't account for the price adjustments that the split or dividend causes. To replace your indicator history with the newly-adjusted prices, call the `Reset` method and then warm up the indicator with `ScaledRaw` data from a history request .

```C#
// In OnData
if (data.Splits.ContainsKey(_symbol) || data.Dividends.ContainsKey(_symbol))
{
    // Reset the indicator
    _sma.Reset();

    // Warm up the indicator
    var tradeBars = History<TradeBar>(_symbol, _sma.WarmUpPeriod, Resolution.Daily, dataNormalizationMode:
DataNormalizationMode.ScaledRaw);
    foreach (var tradeBar in tradeBars)
    {
        _sma.Update(tradeBar.EndTime, tradeBar.Close);
    }
}
```

Common Design Patterns

The location in your algorithm where you create and manage indicators depends on the type of universe you use.

One Asset In a Static Universe

If you only have one security in your algorithm, you can create indicators for it during initialization .

```C#
private Symbol _symbol;
private SimpleMovingAverage _sma;

public override void Initialize()
{
    _symbol = AddEquity("SPY").Symbol;
    Settings.AutomaticIndicatorWarmUp = true;
    _sma = SMA(_symbol, 20, Resolution.Daily);
}
```

Multiple Assets or a Dynamic Universe

If your algorithm has multiple assets or a dynamic universe of assets, as you receive new `Security` objects in `OnSecuritiesChanged` , cast them to `dynamic` objects and add custom members to them.

```C#
public override void OnSecuritiesChanged(SecurityChanges changes)
{
    foreach (var security in changes.AddedSecurities)
    {
        var dynamicSecurity = security as dynamic;

        // Create an indicator
        dynamicSecurity.Indicator = SMA(security.Symbol, 10);

        // Warm up the indicator
        WarmUpIndicator(security.Symbol, dynamicSecurity.Indicator);
    }

    foreach (var security in changes.RemovedSecurities)
    {
        DeregisterIndicator((security as dynamic).Indicator);
    }
}
```

Differences From Third-Party Indicators

The values of our indicators can sometimes produce different results than the indicators on other platforms. There can be several reasons for these discrepancies.

Price Differences

If you find differences in indicator values, compare the prices that are fed into the indicator on each platform. If the input data is slightly different, the indicator values will be different. To test if it's a difference in price data, feed in the data from the third-party platform into our indicators. We validate the indicators against third-party sources and when the values are the same, the indicator values are similar too.

Timestamp Differences

We timestamp our data to the time when the period ends. Many other platforms timestamp to the beginning of the candle.

Implementation Differences

Some indicators can have slightly different default arguments for their implementation. A common difference across platforms is to use a different `MovingAverageType` for the indicators. To view the default arguments for all our indicators, see Supported Indicators . To view the full implementation of our indicators, see the LEAN GitHub repository .

Warm Up Period Differences

Some platforms use all of the historical data to warm up their indicators while LEAN indicators have a fixed number of bars they need to warm up. As a result, indicators with long memory like the Exponential Moving Average can have slightly different values across platforms.

Risk Free Interest Rate Differences

Some indicators, like Sharpe ratios and Sortino ratios , are a function of the risk free interest rate . LEAN provides several different ways to define the interest rate, which can lead to different indicator values compared to other platforms.

Examples

Demonstration Algorithms
IndicatorSuiteAlgorithm.cs C# EmaCrossUniverseSelectionAlgorithm.cs C# MACDTrendAlgorithm.cs C# RegressionChannelAlgorithm.cs C#

Indicators

Manual Indicators

Introduction

Manual indicators are indicators that don't automatically update from the underlying security data. Manual updates let you customize how you update the indicator, such as using a custom field or using Renko Bars . It's not always necessary to create manual indicators though. Automatic indicators are easier to create and update.

Naming Convention

The class name to create a manual indicator is the indicator name spelled in pascal case. For example, to create a manual simple moving average indicator , use the `SimpleMovingAverage` class.

Create Manual Indicators

To create manual indicators, instantiate an indicator with its constructor. To view all of the available indicators and their constructors, see Supported Indicators .

```C#
// Create an indicator
private Delay _delay;
_delay = new Delay(20);

// Create a candlestick pattern
private TwoCrows _twoCrows;
_twoCrows = new TwoCrows();
```

You can track indicators by their name. To name a manual indicator, pass a string as the first argument to the constructor.

```C#
// Name an indicator
_delay = new Delay("AAPL Past Price", 20);
Log(_delay.Name);

// Name a candlestick pattern
_twoCrows = new TwoCrows("Two Crows Pattern");
Log(_twoCrows.Name);
```

If you don't name an indicator, it's given a default name. For example, the default name for a `Delay` indicator is "Delay(period)".

Manual Updates

With manual updates, you control what data you use to update the indicator. For instance, you can use the 3:30 PM price in your daily moving average instead of the daily closing price or you can use the maximum temperature of the past 10 cloudy days.

To update an indicator, call the `Update` method. The `Update` method expects one of the following arguments:

- Time/decimal pair
- `IndicatorDataPoint`
- `QuoteBar`
- `TradeBar`

- Custom data bar

To view what data type you should use to update an indicator, see Supported Indicators .

You can update indicators at any point in your algorithm, but the most common places are during the `OnData` event handler or during a consolidation event handler.

```
C#

private Symbol _symbol;
private RelativeStrengthIndex _rsi;
private AccumulationDistribution _ad;

public override void Initialize()
{
    _symbol = AddEquity("SPY", Resolution.Daily).Symbol;

    _rsi = new RelativeStrengthIndex(10, MovingAverageType.Simple);
    _ad = new AccumulationDistribution();

    var consolidator = new TradeBarConsolidator(TimeSpan.FromDays(3));   // 10 x 3 daily trade bars to warm up the
RSI.
    consolidator = new QuoteBarConsolidator(TimeSpan.FromDays(3));       // 10 x 3 daily quote bars to warm up the
RSI.
    consolidator = new RenkoBarConsolidator(1);     // Renko consolidator that emits a bar when the price moves $1
    //Update the AD indicator with the consolidated bar
    consolidator.DataConsolidated += (_, bar) => _ad.Update(bar);
    SubscriptionManager.AddConsolidator(_symbol, consolidator);
}

public override void OnData(Slice slice)
{
    // Update the RSI indicator value with the new input close price every day
    if (slice.Bars.ContainsKey(_symbol))
    {
        var bar = slice.Bars[_symbol];
        _rsi.Update(bar.EndTime, bar.Close);
    }
}
```

Automatic Updates

With automatic updates, your indicators automatically update with the security data on a schedule you set. To configure automatic updates, create a consolidator and then call the `RegisterIndicator` method. If you register an indicator for automatic updates, don't call the indicator's `Update` method or else the indicator will receive double updates.

```
C#

// Create a security subscription
_symbol = AddEquity("SPY", Resolution.Hour);

// Create a manual indicator
_indicator = new RelativeStrengthIndex(10, MovingAverageType.Simple);

// Create a consolidator
var consolidator = new TradeBarConsolidator(3);   // 10 x 3 min trade bars to warm up the RSI.
consolidator = new QuoteBarConsolidator(4);       // 10 x 4 min quote bars to warm up the RSI.
consolidator = new RenkoConsolidator(1);          // Renko consolidator that emits a bar when the price moves $1

// Register the indicator to update with the consolidated data
RegisterIndicator(_symbol, _indicator, consolidator);
```

Data point indicators use only a single price data in their calculations. By default, those indicators use the closing price. For assets with `TradeBar` data, that price is the `TradeBar` close price. For assets with `QuoteBar` data, that price is the mid-price of the bid closing price and the ask closing price. To create an indicator with the other fields like the `Open` , `High` , `Low` , or `Close` , provide a `selector` argument to the `RegisterIndicator` method.

```
RegisterIndicator(_symbol, _rsi, consolidator, Field.High);
```

The `Field` class has the following `selector` properties:

To create a custom `selector` , define a function that calculates the value.

```
RegisterIndicator(_symbol, _indicator, _consolidator, x =>
{
    var bar = x as IBaseDataBar;
    return (bar.Low + bar.High) / 2;
});
```

To stop automatically updating an indicator, pass the indicator to the `DeregisterIndicator` property method.

```
DeregisterIndicator(_indicator);
// Alias:
// UnregisterIndicator(_indicator);
```

Warm Up Indicators

Indicators use historical data to compute their value. Before you start trading with an indicator, warm it up. There are several ways to warm-up manual indicators.

Manual Indicator Warm-up

If you have access to the `QCAlgorithm` object, you can manually warmup indicators with a history request .

```
private SimpleMovingAverage _sma;
// Create a manual indicator.
_sma = SimpleMovingAverage(20);
// Get 20 bars of daily data.
var history = algorithm.History(_symbol, 20, Resolution.Daily);
// Manually warm up the indicator with the historical close prices.
foreach (var bar in history)
{
    sma.Update(bar.Time, bar.Close);
}
```

Warm-up Helper Method

If an indicator inherits (implements) the `IIndicatorWarmUpPeriodProvider` interface, you can warm it up with the `WarmUpIndicator` method.

```
// Warm up the indicator using the warm-up helper.
_sma = SimpleMovingAverage(20);
algorithm.WarmUpIndicator(_symbol, _sma);
```

To warm up the indicator with a resolution that's different from the security resolution, pass a resolution or `TimeSpan` argument to the `WarmUpIndicator` method. The resolution you provide should be greater than or equal to the security resolution. For example, if the security has minute resolution data, you should warm up the indicator with data that spans at least one minute.

```
// Warm-up with daily bars.
algorithm.WarmUpIndicator(_symbol, _sma, Resolution.Daily);

// Warm-up with 3-day bars.
algorithm.WarmUpIndicator(_symbol, _sma, TimeSpan.FromDays(3));
```

The `WarmUpIndicator` method uses the default `Value` of the historical data to warm up the indicator. In most cases, this is the closing price. To warm up the indicator with an alternative price field, pass a `Field` argument to the method.

```
// Warm up the indicator using daily high prices instead of close prices.
algorithm.WarmUpIndicator(_symbol, _sma, Resolution.Daily, Field.High);
```

Some indicators require the prices of two assets to compute their value (for example, Beta). In this case, pass a list of the `Symbol` objects to the method.

```
// Use the warm-up helper for an indicator that needs the data of 2 assets.
var spy = AddEquity("SPY").Symbol;
var aapl = AddEquity("AAPL").Symbol;
var beta = new Beta("", aapl, spy, 21);
WarmUpIndicator(new[] {aapl, spy}, beta, Resolution.Daily);
```

Algorithm Warm-up

If you create indicators at the beginning of your algorithm, you can set an algorithm warm-up period to warm up the indicators. When you set an algorithm warm-up period, the engine pumps data in and automatically updates all the indicators from before the start date of the algorithm. To ensure that all the indicators are ready after the algorithm warm-up period, choose a lookback period that contains sufficient data.

```
private SimpleMovingAverage _sma;

// In the Initialize method, request data and construct the indicators you need.
var symbol = AddEquity("SPY", Resolution.Daily).Symbol;
_sma = SimpleMovingAverage(20);
SetWarmUp(20);

// In the OnData event handler, update your indicators.
// Use the IsWarmingUp flag to skip some code execution.
_sma.Update(data["SPY"]); // Delete this line if you registered the indicator for automatic updates.
if (IsWarmingUp)
{
    return;
}
```

Timing Considerations

In some cases, you might create and warm up the indicator during its sampling period. For example, say the security resolution is minute, the indicator resolution is daily, and you create and warm-up the indicator at noon without using the `SetWarmUp` method. In this example, the history request that gathers data to warm up the indicator won't contain any data from the current day. Furthermore, if you set up a consolidator to update the indicator, the consolidator also won't aggregate any data from before noon. It doesn't cause issues if the indicator only uses the close price to calculate the indicator value (like the simple moving average indicator) because the first consolidated bar that updates the indicator will have the correct close price. However, if the indicator uses more than just the close price to calculate its value (like the True Range indicator), the open, high, and low values of the first consolidated bar may be incorrect, causing the initial indicator values to be incorrect.

Examples

The following examples demonstrate some common practices for using manual indicators.

Example 1: Universe Selection

The following algorithm selects the stocks within 5% of the 1-year maximum price among the top 100 liquid stocks. To do so, we make use of Maximum indicator to do so. Then, we hold the stocks with price > EMA > SMA , which indicates an upward accelerating trend.

```C#
public class ManualIndicatorAlgorithm : QCAlgorithm
{
    private Dictionary<Symbol, Maximum> _maximumBySymbol = new();
    private Dictionary<Symbol, SymbolData> _symbolData = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Select a popularity-based universe with indicators in a Selection function.
        AddUniverse(Selection);

        // Set a scheduled event to rebalance daily on the daily indicator signals.
        Schedule.On(
            DateRules.EveryDay(),
            TimeRules.At(9, 31),
            Rebalance
        );
    }

    private IEnumerable<Symbol> Selection(IEnumerable<Fundamental> fundamentals)
    {
        var selected = new List<Symbol>();

        // Initially filtered for the top 100 liquid stocks first.
        var filtered = fundamentals.OrderByDescending(f => f.DollarVolume)
            .Take(100)
            .ToList();

        foreach (var f in filtered)
        {
            if (!_maximumBySymbol.TryGetValue(f.Symbol, out var maximum))
            {
                maximum = new(252);
                // Warm up the Maximum indicator for its readiness to use immediately.
                var history = History<TradeBar>(f.Symbol, 252, Resolution.Daily);
                foreach (var bar in history)
                {
                    maximum.Update(bar.EndTime, bar.Close);
                }
                _maximumBySymbol[f.Symbol] = maximum;
            }
            else
            {
                // Update the indicator with the last known adjusted price daily.
                maximum.Update(f.EndTime, f.AdjustedPrice);
            }

            // Select to trade if the current price is within 5% of the maximum price of the last year.
            // Close to the maximum price provides evidence of high popularity for the fund to support the trend.
            if (f.AdjustedPrice >= maximum * 0.95m)
            {
                selected.Add(f.Symbol);
            }
        }

        return selected;
    }

    private void Rebalance()
    {
        var symbolsToBuy = new List<Symbol>();

        foreach (var (symbol, symbolData) in _symbolData)
        {
```

```
        {
            // Buy the stocks whose prices are above the EMA and the EMA is above the SMA, meaning their trend is
upward accelerating.
            if (symbolData.IsReady && Securities[symbol].Price > symbolData.Ema && symbolData.Ema > symbolData.Sma)
            {
                symbolsToBuy.Add(symbol);
            }
        }

        // Equally invest in selected stocks to dissipate capital risk evenly.
        var count = symbolsToBuy.Count;
        if (count > 0)
        {
            var targets = symbolsToBuy.Select(symbol => new PortfolioTarget(symbol, 1m / count)).ToList();
            // Liquidate the positions that are not on an upward trend or are not popular anymore.
            SetHoldings(targets, liquidateExistingHoldings: true);
        }
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach (var added in changes.AddedSecurities)
        {
            // Instantiate a new instance of SymbolData object to hold indicators to use.
            _symbolData[added.Symbol] = new SymbolData(this, added.Symbol);
        }

        foreach (var removed in changes.RemovedSecurities)
        {
            // Remove the data subscription of indicators to release computation resources when leaving the universe.
            if (_symbolData.Remove(removed.Symbol, out var symbolData))
            {
                symbolData.Dispose();
            }
        }
    }

    private class SymbolData
    {
        private QCAlgorithm _algorithm;
        public Symbol Symbol { get; set; }
        public ExponentialMovingAverage Ema { get; set; }
        public SimpleMovingAverage Sma { get; set; }

        public bool IsReady => Ema.IsReady && Sma.IsReady;

        public SymbolData(QCAlgorithm algorithm, Symbol symbol)
        {
            _algorithm = algorithm;
            Symbol = symbol;

            // Create an EMA and an SMA manual indicator for trade signal generation.
            Ema = new ExponentialMovingAverage(20);
            Sma = new SimpleMovingAverage(20);

            // Warm up the indicators to ensure their readiness to use them immediately.
            algorithm.WarmUpIndicator(symbol, Ema, Resolution.Daily);
            algorithm.WarmUpIndicator(symbol, Sma, Resolution.Daily);

            // Subscribe to the indicators to update daily price data for updated trade signal generation.
            algorithm.RegisterIndicator(symbol, Ema, Resolution.Daily);
            algorithm.RegisterIndicator(symbol, Sma, Resolution.Daily);
        }

        public void Dispose()
        {
            Ema.Reset();
            Sma.Reset();
            // Cancel data subscription to release computation resources.
            _algorithm.DeregisterIndicator(Ema);
            _algorithm.DeregisterIndicator(Sma);
        }
    }
}
```

Indicators

Automatic Indicators

Introduction

Automatic indicators are indicators that automatically update from the underlying security data. Automatic indicators do everything that manual indicators do, but with automatic updates registered for you.

Naming Convention

The method to create an automatic indicator is usually named after the acronym of the indicator name. For example, to create an automatic simple moving average indicator , use the `SMA` method.

Create Automatic Indicators

To create automatic indicators, call the indicator helper method from the `QCAlgorithm` class. The indicator helper methods create an indicator object, hooks it up for automatic updates, and returns it so you can used it in your algorithm. To view the helper method for each indicator, see Supported Indicators . In most cases, you should call the helper method in the `Initialize` method.

The indicator resolution must be greater than or equal to the resolution of the security subscription. For instance, if your security subscription is for minute resolution data, the indicator resolution should be minute, hour, or daily resolution.

```C#
private SimpleMovingAverage _sma;
private TwoCrows _twoCrows;

// In Initialize()
var symbol = AddEquity("SPY").Symbol;

//  - Create an indicator
_sma = SMA(symbol, 20, Resolution.Daily);

// - Create a candlestick pattern
var patterns = new CandlestickPatterns(this);
_twoCrows = patterns.TwoCrows(_symbol);
```

When you create an indicator with a helper method, the indicator is given a default name. You can track indicators by their name. The default name for the `SimpleMovingAverage` indicator is " SMA (period , ticker _ resolution)". For example, in the preceding code snippet, the `SimpleMovingAverage` indicator is named " SMA (20,SPY_day)". To get the name of an indicator, inspect its `Name` property.

```C#
Log(_sma.Name);
Log(_twoCrows.Name);
```

Alternative Price Fields

Data point indicators use only a single price data in their calculations. By default, those indicators use the closing price. For assets with `TradeBar` data, that price is the `TradeBar` close price. For assets with `QuoteBar` data, that price is the mid-price of the bid closing price and the ask closing price. To create an indicator with the other fields like the `Open` , `High` , `Low` , or `Close` , provide a `selector` argument to the indicator helper method.

```
// Define a 10-period daily RSI indicator with shortcut helper method
// Select the Open price to update the indicator
_rsi = RSI("SPY", 10,  MovingAverageType.Simple, Resolution.Daily, Field.Open);
```

The `Field` class has the following `selector` properties:

To create a custom `selector` , define a function that calculates the value.

```
_rsi = RSI("SPY", 10,  MovingAverageType.Simple, Resolution.Daily, x =>
{
    var bar = x as IBaseDataBar;
    return (bar.Low + bar.High) / 2;
});
```

Warm Up Indicators

Indicators compute their value based on historical data. Before you start trading with an indicator, warm it up. There are several ways to warm-up automatic indicators.

Algorithm Warm-up

You can set an algorithm warm-up period to warm up the indicators. When you set an algorithm warm-up period, the engine pumps data in and automatically updates all the indicators from before the start date. To ensure that all the indicators are ready after the algorithm warm-up period, choose a look-back period that contains sufficient data.

```
private SimpleMovingAverage _sma;

// In Initialize
var symbol = AddEquity("SPY").Symbol;
_sma = SMA(symbol, 20, Resolution.Daily);
SetWarmUp(20, Resolution.Daily);

// In OnData
if (IsWarmingUp)
{
    return;
}
```

Manual Indicator Warm-up

You can manually warm up indicators with a history request .

```
private SimpleMovingAverage _sma;

// In Initialize
var symbol = AddEquity("SPY").Symbol;
_sma = SMA(symbol, 20, Resolution.Daily);

var history = History(symbol, 20, Resolution.Daily);
foreach (var bar in history)
{
    sma.Update(bar.Time, bar.Close);
}
```

Automatic Indicator Warm-up

You can set the `Settings.AutomaticIndicatorWarmUp` property to true before you create most indicators to automatically warm

them up. This technique doesn't work for all indicators.

```
C#
private SimpleMovingAverage _sma;

// In Initialize
var symbol = AddEquity("SPY").Symbol;
Settings.AutomaticIndicatorWarmUp = true;
_sma = SMA(symbol, 20, Resolution.Daily);
```

Warm-up Helper Method

If an indicator inherits the `IIndicatorWarmUpPeriodProvider` class, you can warm it up with the `WarmUpIndicator` method.

```
C#
_sma = SMA(_symbol, 20);
WarmUpIndicator(_symbol, _sma);
```

To warm up the indicator with a resolution that's different from the security resolution, pass a resolution or `TimeSpan` argument to the `WarmUpIndicator` method. The resolution you provide should be greater than or equal to the security resolution. For example, if the security has minute resolution data, you should warm up the indicator with data that spans at least one minute. If the indicator resolution is different from the security resolution, provide the indicator resolution as the argument to properly warm up the indicator.

```
C#
// Warm up with daily bars
WarmUpIndicator(_symbol, _sma, Resolution.Daily);

// Warm up with 3-day bars
WarmUpIndicator(_symbol, _sma, TimeSpan.FromDays(3));
```

The `WarmUpIndicator` method uses the default `Value` of the historical data to warm up the indicator. In most cases, this is the closing price. To warm up the indicator with a different data field, pass a `Field` argument to the method.

```
C#
WarmUpIndicator(_symbol, _sma, Resolution.Daily, Field.High);
```

Some indicators require the prices of two assets to compute their value (for example, Beta). In this case, pass a list of the `Symbol` objects to the method.

```
C#
var spy = AddEquity("SPY").Symbol;
var aapl = AddEquity("AAPL").Symbol;
var beta = B(aapl, spy, 21, Resolution.Daily);
WarmUpIndicator(new[] {aapl, spy}, beta, Resolution.Daily);
```

Timing Considerations

In some cases, you might create and warm up the indicator during its sampling period. For example, say the security resolution is minute, the indicator resolution is daily, and you create and warm-up the indicator at noon without using the `SetWarmUp` method. In this example, the history request that gathers data to warm up the indicator won't contain any data from the current day and the consolidator that updates the indicator also won't aggregate any data from before noon. It doesn't cause issues if the indicator only uses the close price to calculate the indicator value (like the simple moving average indicator) because the first consolidated bar that updates the indicator will have the correct close price. However, if the indicator uses more than just

the close price to calculate its value (like the True Range indicator), the open, high, and low values of the first consolidated bar may be incorrect, causing the initial indicator values to be incorrect.

Deregister Indicators

To stop the automatic updates of an indicator, pass the indicator to the `DeregisterIndicator` property method.

```
C#

DeregisterIndicator(_sma);
// Alias:
// UnregisterIndicator(_sma);
```

Common Mistakes

Avoid the following common mistakes when you use automatic indicators.

Using Indicator Values Before the Indicator Is Warmed Up

Indicators can provide inaccurate output values before they are warmed up. To avoid issues, always check the `IsReady` flag before you use indicator output values.

```
C#

// IsReady signals whether the indicator values are accurate and ready to use.
if (_indicator.IsReady)
{
    var value = _indicator.Current.Value;
}
```

Manually Registering an Automatic Indicator for Updates

If you create an automatic indicator and then register it for automatics updates or call the `Update` method, the indicator receives multiple input values during each update cycle. To avoid issues, create automatic indicators but don't register them for automatic updates or call the `Update` method.

```
C#

// Automatic indicator short-cuts create the indicator and register it for data updates.
_indicator = SMA(_symbol, 5);

// Don't register the automatic indicators for data. They automatically get data.
// RegisterIndicator(_symbol, _indicator, Resolution.Daily)
```

Examples

The following examples demonstrate some common practices for using automatic indicators.

Example 1: Trade With Automatic Indicators in a Dynamic Universe

The following algorithm selects a universe of the 20 most liquid US Equities each day. At the start of each trading session, it then forms an equal-weighted portfolio of the stocks that have price > EMA > SMA , which indicates an upward accelerating trend. As assets enter the universe, it creates automatic indicators for them. As assets leave the universe, it deregisters the indicators.

```
C#

public class AutomaticIndicatorAlgorithm : QCAlgorithm
{
    private Dictionary<Symbol, SymbolData> _symbolData = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
```

```
        // Add a universe of the 20 most liquid stocks to trade trends more efficiently.
        AddUniverse(Universe.Top(20));

        // Set a Scheduled Event to rebalance at the start of each day with the daily indicator signals.
        Schedule.On(DateRules.EveryDay(), TimeRules.At(9, 31), Rebalance);
    }

    private void Rebalance()
    {
        // Buy the stocks where price is above EMA and EMA is above SMA, meaning its trend is upward and
accelerating.
        var symbolsToBuy = _symbolData.Where(
                kvp => kvp.Value.IsReady && Securities[kvp.Key].Price > kvp.Value.Ema && kvp.Value.Ema >
kvp.Value.Sma
            )
            .Select(kvp => kvp.Key)
            .ToList();

        // Equally invest in the selected stocks to evenly distribute the capital risk.
        var count = symbolsToBuy.Count;
        if (count > 0)
        {
            var targets = symbolsToBuy.Select(symbol => new PortfolioTarget(symbol, 1m / count)).ToList();
            // Liquidate the positions that are no longer popular or in an uptrend.
            SetHoldings(targets, liquidateExistingHoldings: true);
        }
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach (var added in changes.AddedSecurities)
        {
            // Create a SymbolData object for this asset, which is an object that contains the indicators.
            _symbolData[added.Symbol] = new SymbolData(this, added.Symbol);
        }

        foreach (var removed in changes.RemovedSecurities)
        {
            // Stop updating the indicators of assets that leave the universe to release computation resources.
            if (_symbolData.Remove(removed.Symbol, out var symbolData))
            {
                symbolData.Dispose();
            }
        }
    }

    private class SymbolData
    {
        private QCAlgorithm _algorithm;
        public Symbol Symbol { get; set; }
        public ExponentialMovingAverage Ema { get; set; }
        public SimpleMovingAverage Sma { get; set; }

        public bool IsReady => Ema.IsReady && Sma.IsReady;

        public SymbolData(QCAlgorithm algorithm, Symbol symbol)
        {
            _algorithm = algorithm;
            Symbol = symbol;

            // Create EMA & SMA indicators to generate trade signals.
            Ema = algorithm.EMA(symbol, 20, Resolution.Daily);
            Sma = algorithm.SMA(symbol, 20, Resolution.Daily);

            // Warm up the indicators so they are ready to use immediately.
            algorithm.WarmUpIndicator(symbol, Ema, Resolution.Daily);
            algorithm.WarmUpIndicator(symbol, Sma, Resolution.Daily);
        }

        public void Dispose()
        {
            // Stop updating the indicators of assets that leave the universe to release computation resources.
            _algorithm.DeregisterIndicator(Ema);
            _algorithm.DeregisterIndicator(Sma);
        }
    }
}
```

Indicators

Plotting Indicators

Introduction

LEAN provides helper methods to make it simple to create indicator plots.

Plot Update Events

To plot all of the values of some indicators, in the `Initialize` method, call the `PlotIndicator` method. The method plots each indicator value as the indicator updates. The method accepts up to four indicators.

```C#
var symbol = AddEquity("SPY");
var smaShort = SMA(symbol, 10);
var smaLong = SMA(symbol, 20);
PlotIndicator("<chartName>", smaShort, smaLong)
```

Plot Current Values

To plot the current value of indicators, call the `Plot` method. The method accepts up to four indicators.

```C#
// In Initialize
var symbol = AddEquity("SPY");
var smaShort = SMA(symbol, 10);
var smaLong = SMA(symbol, 20);

// In OnData
Plot("<chartName>", smaShort, smaLong)
```

View Charts

The following table describes where you can access your charts, depending on how to deploy your algorithms:

Location	Algorithm Lab Algorithms	CLI Cloud Algorithms	CLI Local Algorithms
Backtest results page	✓	✓	
Live results page	✓	✓	
/backtests/read endpoint	✓	✓	
/live/read endpoint	✓	✓	
ReadBacktest method	✓	✓	
ReadLiveAlgorithm method	✓	✓	
Local JSON file in your <projectName> / backtests / <timestamp> or <projectName> / live / <timestamp> directory		✓	✓

Chart Limits

Not all indicators share the same base type(T), so you may not be able to plot them together since some indicators require points while others require `TradeBars` .

```
// Plot indicators that extend the "Indicator" type
PlotIndicators("All Indicator Values", sma, rsi);
Plot("Current Indicator Values", sma, rsi);

// Plot indicators that extend the "TradeBarIndicator" type
PlotIndicators("All Indicator Values", atr, aroon);
Plot("Current Indicator Values", atr, aroon);
```

If your indicator plots are complex, call the `Plot` method with one indicator and plot its `Current.Value` property. For more information about plotting, see Charting .

Examples

The following examples demonstrate some common practices for plotting indicators.

Example 1: SMA Ribbon

The following algorithm trades simple moving average (SMA) crossings, which involves five SMA indicators to form an SMA ribbon. To visualize the ribbon, it plots each indicator with a different color.

```
public class PlottingIndicatorAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private Dictionary<int, SimpleMovingAverage> _smas;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Add SPY data to update indicators and trade.
        _spy = AddEquity("SPY", Resolution.Daily).Symbol;

        // Create a chart to plot the ribbon. To better visualize the plot, use different colors per series.
        var chart = new Chart("Ribbon");
        chart.AddSeries(new Series("Price", SeriesType.Candle, "$"));
        chart.AddSeries(new Series("SMA10", SeriesType.Line, "$", Color.Violet));
        chart.AddSeries(new Series("SMA20", SeriesType.Line, "$", Color.Blue));
        chart.AddSeries(new Series("SMA30", SeriesType.Line, "$", Color.Green));
        chart.AddSeries(new Series("SMA40", SeriesType.Line, "$", Color.Yellow));
        chart.AddSeries(new Series("SMA50", SeriesType.Line, "$", Color.Red));

        // Create 5 SMA indicators to create the SMA ribbon.
        _smas = new[]{10, 20, 30, 40, 50}.ToDictionary(k => k, v =>
        {
            var sma = SMA(_spy, v, Resolution.Daily);
            // Plot the indicator value to create the ribbon when the indicator updates.
            sma.Updated += (_, point) => Plot("Ribbon", $"SMA{v}", point.Value);
            // Warm up the indicator so it's ready to use immediately.
            WarmUpIndicator(_spy, sma);
            return sma;
        });
    }

    public override void OnData(Slice slice)
    {
        if (slice.Bars.TryGetValue(_spy, out var bar))
        {
            // Plot the candlestick when you get the daily bar.
            Plot("Ribbon", "Price", bar.Open, bar.High, bar.Low, bar.Close);

            // Follow strong upward trend.
            if (IsUptrend(bar.Close, _smas))
            {
```

```
                SetHoldings(_spy, 1m);
            }
            // Follow strong downward trend.
            else if (IsDowntrend(bar.Close, _smas))
            {
                SetHoldings(_spy, -1m);
            }
            // Liquidate positions if the trend is less deterministic.
            else
            {
                Liquidate();
            }
        }
    }

    public bool IsUptrend(decimal price, IDictionary<int, SimpleMovingAverage> smas)
    {
        // Uptrend occurs when price is above all SMAs and shorter SMAs are above longer SMAs.
        var smasList = smas.OrderBy(kvp => kvp.Key)
            .Select(kvp => kvp.Value)
            .ToList();
        return Enumerable.Range(1, smasList.Count-1).All(i => smasList[i] < smasList[i-1])
            && price > smasList[0];
    }

    public bool IsDowntrend(decimal price, IDictionary<int, SimpleMovingAverage> smas)
    {
        // Downtrend occurs when price is below all SMAs and shorter SMAs are below longer SMAs.
        var smasList = smas.OrderBy(kvp => kvp.Key)
            .Select(kvp => kvp.Value)
            .ToList();
        return Enumerable.Range(1, smasList.Count-1).All(i => smasList[i] > smasList[i-1])
            && price < smasList[0];
    }
}
```

Indicators

Combining Indicators

Introduction

Indicator extensions let you chain indications together like Lego blocks to create unique combinations. When you chain indicators together, the `Current.Value` property output of one indicator is the input of the following indicator. To chain indicators together with values other than the `Current.Value` property, create a custom indicator .

Addition

The `Plus` extension sums the `Current.Value` property of two indicators or sums the `Current.Value` property of an indicator and a fixed value.

```C#
// Sum the output of two indicators
var min = MIN("SPY", 21);
var std = STD("SPY", 21);
var minPlusStd = min.Plus(std);

// Sum the output of an indicator and a fixed value
var rsiPlusValue = min.Plus(10);
```

If you pass an indicator to the `Plus` extension, you can name the composite indicator.

```C#
var namedIndicator = min.Plus(std, "Buy Zone");
```

Subtraction

The `Minus` extension subtracts the `Current.Value` property of two indicators or subtracts a fixed value from the `Current.Value` property of an indicator.

```C#
// Subtract the output of two indicators
var smaShort = SMA("SPY", 14);
var smaLong = SMA("SPY", 21);
var smaDifference = smaShort.Minus(smaLong);

// Subtract a fixed value from the output of an indicator
var smaMinusValue = smaShort.Minus(10);
```

If you pass an indicator to the `Minus` extension, you can name the composite indicator.

```C#
var namedIndicator = smaShort.Minus(smaLong, "SMA Difference");
```

Multiplication

The `Times` extension multiplies the `Current.Value` property of two indicators or multiplies a fixed value and the `Current.Value` property of an indicator.

```
// Multiply the output of two indicators
var emaShort = EMA("SPY", 14);
var emaLong = EMA("SPY", 21);
var emaProduct = emaShort.Times(emaLong);

// Multiply the output of an indicator and a fixed value
var emaTimesValue = emaShort.Times(10);
```

If you pass an indicator to the `Times property` extension, you can name the composite indicator.

```
var namedIndicator = emaShort.Times(emaLong, "EMA Product");
```

Division

The `Over` extension divides the `Current.Value` property of an indicator by the `Current.Value` property of another indicator or a fixed value.

```
// Divide the output of two indicators
var rsiShort = RSI("SPY", 14);
var rsiLong = RSI("SPY", 21);
var rsiDivision = rsiShort.Over(rsiLong);

// Divide the output of an indicator by a fixed value
var rsiAverage = rsiShort.Plus(rsiLong).Over(2);
```

If you pass an indicator to the `Over` extension, you can name the composite indicator.

```
var namedIndicator = rsiShort.Over(rsiLong, "RSI Division");
```

Weighted Average

The `WeightedBy` extension calculates the average `Current.Value` property of an indicator over a lookback period, weighted by another indicator over the same lookback period. The value of the calculation is

$$\frac{\textbf{x} \cdot \textbf{y}}{ \sum\limits_{i=1}^{n} y_{i} }$$

where $\textbf{x}$ is a vector that contains the historical values of the first indicator,$\textbf{y}$ is a vector that contains the historical values of the second indicator, and n is the lookback period.

```
var smaShort = SMA("SPY", 14);
var smaLong = SMA("SPY", 21);
var weightedSMA = smaShort.WeightedBy(smaLong, 3);
```

Custom Chains

The `Of` extension feeds an indicator's `Current.Value` property into the input of another indicator. The first argument of the `IndicatorExtensions.Of` method must be a manual indicator with no automatic updates. If you pass an indicator that has automatic updates as the argument, that first indicator is updated twice. The first update is from the security data and the second update is from the `IndicatorExtensions` class.

```
var rsi = RSI("SPY", 14);
var rsiSMA = (new SimpleMovingAverage(10)).Of(rsi); // 10-period SMA of the 14-period RSI
```

If you pass a manual indicator as the second argument, to update the indicator chain, update the second indicator. If you call the `Update` method of the entire indicator chain, it won't update the chain properly.

Simple Moving Average

The `SMA` extension calculates the simple moving average of an indicator's `Current.Value` property.

```
var rsi = RSI("SPY", 14); // Create a RSI indicator
var rsiSMA = rsi.SMA(3); // Create an indicator to calculate the 3-period SMA of the RSI indicator
```

Exponential Moving Average

The `EMA` extension calculates the exponential moving average of an indicator's `Current.Value` property.

```
var rsi = RSI("SPY", 14); // Create a RSI indicator
var rsiEMA = _rsi.EMA(3); // Create an indicator to calculate the 3-period EMA of the RSI indicator
```

The `EMA` extension can also accept a smoothing parameter that sets the percentage of data from the previous value that's carried into the next value.

```
var rsiEMA = rsi.EMA(3, 0.1m); // 10% smoothing factor
```

Maximum

The `MAX` extension calculates an indicator's maximum `Current.Value` property over a lookback window.

```
var ema = EMA("SPY", 14); // Create an EMA indicator
var emaMax = ema.MAX(10); // Create an indicator to calculate the maximum EMA over the last 10 periods
```

Minimum

The `MIN` extension calculates an indicator's minimum `Current.Value` property over a lookback window.

```
var ema = EMA("SPY", 14); // Create an EMA indicator
var emaMin = ema.MIN(10); // Create an indicator to calculate the minimum EMA over the last 10 periods
```

Examples

The following examples demonstrate some common practices for combining indicators.

Example 1: Volatility

The following algorithm trades a volatility strategy. By comparing SMA and the current value of the standard deviation of the return, we can estimate the current volatility regime is above or below average to trade the price volatility through strangle.

C#

```
public class CombiningIndicatorsAlgorithm : QCAlgorithm
{
    private Symbol _spy, _option;
    private SimpleMovingAverage _sma;
    private StandardDeviation _sd;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request daily SPY data to feed the indicators and generate trade signals.
        // Use Raw data normalization mode to compare the strike price fairly.
        _spy = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;

        // Request option data to trade.
        var option = AddOption(_spy);
        _option = option.Symbol;
        // Filter for 7-day expiring options with $5 apart from the current price to trade volatility using strangle.
        option.SetFilter((universe) => universe.IncludeWeeklys().Strangle(7, 5, -5));

        // Create a return indicator to get the daily return of SPY.
        var ret = ROC(_spy, 1, Resolution.Daily);
        // Create an SD indicator to measure the 252-day SD of return to measure SPY's volatility.
        _sd = IndicatorExtensions.Of(new StandardDeviation(252), ret);
        // Create a 20-day SMA indicator of the SD indicator to compare the average volatility.
        _sma = IndicatorExtensions.Of(new SimpleMovingAverage(20), _sd);

        // Warm up for immediate usage of indicators.
        SetWarmUp(400, Resolution.Daily);
    }

    public override void OnData(Slice slice)
    {
        if (IsWarmingUp)
        {
            return;
        }
        if (!Portfolio.Invested && slice.OptionChains.TryGetValue(_option, out var chain))
        {
            // Create a strangle strategy to trade the volatility forecast.
            var otmCallStrike = chain.Max(x => x.Strike);
            var otmPutStrike = chain.Min(x => x.Strike);
            var expiry = chain.Min(x => x.Expiry);
            var strangle = OptionStrategies.Strangle(_option, otmCallStrike, otmPutStrike, expiry);

            // If the current STD is above its SMA, we estimate the volatility will remain high due to volatility
clustering.
            // Thus, we long the strangle to earn from the price displacement from the current level.
            if (_sd > _sma)
            {
                Buy(strangle, 2);
            }
            // If the current STD is below its SMA, we estimate the volatility will remain lower due to volatility
clustering.
            // Thus, we short the strangle to earn from the price staying at the current level.
            if (_sd < _sma)
            {
                Sell(strangle, 2);
            }
        }
        else if (Portfolio[_spy].Invested)
        {
            // Liquidate any assigned underlying positions.
            Liquidate(_spy);
        }
    }
}
```

Example 2: Displaced SMA Ribbon

The following algorithm trades trends indicated by SMA crossings. We use the

`IndicatorExtensions to confirm the trend better.Of` method to create a Delay indicator on SMA indicator.

C#

```
public class DisplacedMovingAverageRibbon : QCAlgorithm
{
    private Symbol _spy;
    private IndicatorBase<IndicatorDataPoint>[] _ribbon;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request daily SPY data for feeding indicator and trading.
        _spy = AddEquity("SPY", Resolution.Daily).Symbol;

        // Create 6 15-day SMA indicators, with a 5-day delay between each indicator.
        const int count = 6;
        const int offset = 5;
        const int period = 15;

        // Define our sma as the base of the ribbon.
        var sma = new SimpleMovingAverage(period);

        _ribbon = Enumerable.Range(0, count).Select(x =>
        {
            // Define our offset to the zero SMA; these various offsets will create our 'displaced' ribbon.
            var delay = new Delay(offset*(x+1));
            // Using Delay indicator to create displaced SMA indicators.
            var delayedSma = delay.Of(sma);
            // Register our new 'delayed_sma' for automatic updates on a daily resolution.
            RegisterIndicator(_spy, delayedSma, Resolution.Daily, data => data.Value);
            return delayedSma;
        }).ToArray();
    }

    public override void OnData(Slice slice)
    {
        // Trade only on updated data with ready-to-use indicators.
        if (!_ribbon.All(x => x.IsReady)) return;
        if (!slice.Bars.TryGetValue(_spy, out var data))
        {
            return;
        }
        // Plot indicators each time they update.
        Plot("Ribbon", "Price", data.Price);
        Plot("Ribbon", _ribbon);

        var values = _ribbon.Select(x => x.Current.Value).ToArray();
        var holding = Portfolio[_spy];
        // Buy SPY if the trend is upward.
        if (holding.Quantity <= 0 && IsAscending(values))
        {
            SetHoldings(_spy, 1.0);
        }
        // Liquidate if the trend is downwards.
        else if (holding.Quantity > 0 && IsDescending(values))
        {
            Liquidate(_spy);
        }
    }

    // Returns true if the SMA values are in ascending order, indicating an upward trend
    private bool IsAscending(IEnumerable<decimal> values)
    {
        decimal? last = null;
        foreach (var val in values)
        {
            if (last == null)
            {
                last = val;
                continue;
            }

            if (last.Value < val)
            {
                return false;
            }
            last = val;
        }
        return true;
    }

    // Returns true if the SMA values are in descending order, indicating a downward trend
```

```
    // Returns true if the SMA values are in descending order, indicating a downward trend
    private bool IsDescending(IEnumerable<decimal> values)
    {
        decimal? last = null;
        foreach (var val in values)
        {
            if (last == null)
            {
                last = val;
                continue;
            }

            if (last.Value > val)
            {
                return false;
            }
            last = val;
        }
        return true;
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm

Indicators

Custom Indicators

Introduction

LEAN supports over 100 pre-built indicators, but a custom indicator is an indicator you define. It receives input, performs some calculation, and sets its output value. Custom indicators are helpful when you want to achieve any of the following results:

- Use an indicator that LEAN doesn't currently implement
- Combine built-in indicators beyond the logic of Indicator Extensions
- Create your own unique indicator for trading

Define Indicators

Custom indicators subsclass the `IndicatorBase<IndicatorDataPoint>` , `BarIndicator` , or `TradeBarIndicator` class, depending on the indicator type . The following definition provides an example of a custom simple moving average indicator that inherits the `IndicatorBase<IndicatorDataPoint>` class. To view examples of indicators that inherit the `BarIndicator` or `TradeBarIndicator` class, see the AverageTrueRange or VolumeWeightedAveragePriceIndicator implementation in the LEAN GitHub repository.

```C#
public class CustomSimpleMovingAverage : IndicatorBase<IndicatorDataPoint>, IIndicatorWarmUpPeriodProvider
{
    private RollingWindow<decimal> _window;
    public override bool IsReady => _window.IsReady;
    public int WarmUpPeriod => _window.Size;

    public CustomSimpleMovingAverage(string name, int period) : base(name)
    {
        _window = new RollingWindow<decimal>(period);
    }

    protected override decimal ComputeNextValue(IndicatorDataPoint input)
    {
        _window.Add(input.Value);
        return _window.Sum() / _window.Size;
    }

    public override void Reset()
    {
        _window.Reset();
        base.Reset();
    }
}
```

The following definition provides an example of a custom money flow index indicator.

```C#
public class CustomMoneyFlowIndex : TradeBarIndicator, IIndicatorWarmUpPeriodProvider
{
    private decimal _previousTypicalPrice;
    private RollingWindow<decimal> _negativeMoneyFlow;
    private RollingWindow<decimal> _positiveMoneyFlow;
    public override bool IsReady => _positiveMoneyFlow.IsReady;
    public int WarmUpPeriod => _positiveMoneyFlow.Size;

    public CustomMoneyFlowIndex(string name, int period) : base(name)
    {
        _negativeMoneyFlow = new(period);
        _positiveMoneyFlow = new(period);
        _previousTypicalPrice = 0m;
    }

    protected override decimal ComputeNextValue(TradeBar input)
    {
        var typicalPrice = (input.High + input.Low + input.Close) / 3;
        var moneyFlow = typicalPrice * input.Volume;

        _negativeMoneyFlow.Add(typicalPrice < _previousTypicalPrice ? moneyFlow: 0);
        _positiveMoneyFlow.Add(typicalPrice > _previousTypicalPrice ? moneyFlow: 0);
        _previousTypicalPrice = moneyFlow;

        var positiveMoneyFlowSum = _positiveMoneyFlow.Sum();
        var totalMoneyFlow = positiveMoneyFlowSum + _negativeMoneyFlow.Sum();

        return totalMoneyFlow == 0 ? 100m : 100m * positiveMoneyFlowSum / totalMoneyFlow;
    }

    public override void Reset()
    {
        _previousTypicalPrice = 0m;
        _negativeMoneyFlow.Reset();
        _positiveMoneyFlow.Reset();
        base.Reset();
    }
}
```

Create Indicators

You must define a custom indicator before you can create an instance of it.

To create a custom indicator, call the indicator constructor.

```C#
private CustomSimpleMovingAverage _sma;

_sma = new CustomSimpleMovingAverage("My SMA", 10);
```

Updates

The process to update custom indicators is the same process you use to update manual indicators. For more information about updating manual indicators, see Manual Updates or Automatic Updates .

Warm Up Indicators

The process to warm up custom indicators is the same process you use to warm up manual indicators .

Examples

The following examples demonstrate some common practices for implementing custom indicators.

Example 1: Multiple Symbol Custom Indicator

The following algorithm implements a custom Cointegration price divergence indicator involving 2 symbols: GOOGL & GOOG. It trades the arbitrage between 2 cointegrated stocks on their price convergence after deviating more than 2 standard deviations.

C#

```
using Accord.Statistics;
using MathNet.Numerics.LinearRegression;

public class CombiningIndicatorsAlgorithm : QCAlgorithm
{
    private Symbol _goog1, _goog2;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request 2 classes of Google stock data to feed the indicators and generate trade signals.
        var goog1 = AddEquity("GOOGL").Symbol;
        var goog2 = AddEquity("GOOG").Symbol;

        // Create a custom money flow index to generate a trade signal.
        var cointegration = new Cointegration(this, goog1, goog2, 252, DateRules.MonthStart());
        // Add a handler to trade on updates.
        cointegration.Updated += OnUpdated;
        // Register the indicator to update automatically with daily data.
        RegisterIndicator(goog1, cointegration, Resolution.Daily);
        RegisterIndicator(goog2, cointegration, Resolution.Daily);

        // Warm up for immediate usage of indicators.
        WarmUpIndicator(goog1, cointegration, Resolution.Daily);
        WarmUpIndicator(goog2, cointegration, Resolution.Daily);
    }

    private void OnUpdated(object sender, IndicatorDataPoint point)
    {
        var indicator = (sender as Cointegration);
        if (indicator.IsReady)
        {
            var holding = Portfolio[indicator.Symbol1];

            // If the residual is lower than -2x SD, it means class A price is much higher than what it should be
compared to class C.
            // We sell class A and buy class C to bet on their price convergence.
            if (point < -2m && !holding.IsShort)
            {
                SetHoldings(indicator.Symbol1, -0.5m);
                SetHoldings(indicator.Symbol2, 0.5m * indicator.Ratio);
            }
            // If the residual is higher than the threshold, it means class A price is much lower than what it should
be compared to class C.
            // We buy class A and sell class C to bet on their price convergence.
            else if (point > 2m && !holding.IsLong)
            {
                SetHoldings(indicator.Symbol1, 0.5m);
                SetHoldings(indicator.Symbol2, -0.5m * indicator.Ratio);
            }
            // Close positions of the price are converged.
            else if ((holding.IsShort && point > 0m) || (holding.IsLong && point < 0m))
            {
                Liquidate();
            }
        }
    }
}

public class Cointegration : IndicatorBase<IndicatorDataPoint>, IIndicatorWarmUpPeriodProvider
{
    // The standard deviation of the residuals, such that the returned indicator value is relative to the SD.
    private decimal _residualSD = 1e10m;
    // Store the coefficient and intercept of the cointegrated series for calculating the spread of a new data point.
    private decimal[] _coefficients = new[] { 0m, 0m };
    private Dictionary<Symbol, RollingWindow<IndicatorDataPoint>> _windows = new();
    private readonly int _period;

    public Symbol Symbol1 { get; set; }

    public Symbol Symbol2 { get; set; }

    // The ratio of Symbol2 within the ocintegration relationship compared to Symbol1.
    public decimal Ratio => _coefficients[0];

    public int WarmUpPeriod => _period;
```

```
    public override bool IsReady => _windows.All(kvp => kvp.Value.IsReady) && _coefficients.All(x => x != 0m);

    public Cointegration(QCAlgorithm algorithm, Symbol symbol1, Symbol symbol2, int period, IDateRule
recalibratingDateRule)
        : base("Cointegration")
    {
        Symbol1 = symbol1;
        Symbol2 = symbol2;

        // Use rolling windows to save the price data for cointegration analysis.
        _windows[symbol1] = new(period);
        _windows[symbol2] = new(period);

        // Adjust the cointegration factor between the 2 classes' price series.
        algorithm.Schedule.On(
            recalibratingDateRule,
            algorithm.TimeRules.At(23, 59),
            CalculateCointegration
        );

        _period = period;
    }

    public override bool Update(IBaseData input)
    {
        // Update the rolling windows for cointegration analysis.
        if (!_windows.TryGetValue(input.Symbol, out var window))
        {
            throw new ArgumentException($"{input.Symbol} is not part of the Cointegration relation.");
        }
        window.Add(new IndicatorDataPoint(input.Symbol, input.EndTime, input.Value));

        var nextResult = ValidateAndComputeNextValue((IndicatorDataPoint)input);
        if (nextResult.Status == IndicatorStatus.Success)
        {
            Current = new IndicatorDataPoint(input.EndTime, nextResult.Value);
            OnUpdated(Current);
        }

        return IsReady;
    }

    protected override IndicatorResult ValidateAndComputeNextValue(IndicatorDataPoint input)
    {
        // If no cointegration relationship is found, an invalid result will always be returned.
        if (_coefficients.All(x => x == 0m))
        {
            return new IndicatorResult(0m, IndicatorStatus.ValueNotReady);
        }
        return new IndicatorResult(ComputeNextValue(input));
    }

    protected override decimal ComputeNextValue(IndicatorDataPoint input)
    {
        // Calculate the updated cointegrated series spread only if all symbol data points are updated.
        if (_windows.All(kvp => kvp.Value[0].EndTime == input.EndTime))
        {
            return (_coefficients[0] * _windows[Symbol2][0].Value + _coefficients[1] - _windows[Symbol1][0].Value) /
_residualSD;
        }
        return Current.Value;
    }

    private void CalculateCointegration()
    {
        // Lag direction is unimportant; it is just a sign flip in the linear regression, so we don't need to flip
the window order.
        var y = _windows[Symbol1].Select(x => (double)x.Value).ToArray();
        var x = _windows[Symbol2].Select(x => (double)x.Value).ToArray();
        if (x.Length < 2 || y.Length < 2)
        {
            return;
        }

        // Perform Linear Regression on both price series to investigate their relationship.
        var regressionResult = SimpleRegression.Fit(x, y);
        var intercept = regressionResult.Item1;
        var slope = regressionResult.Item2;

        // Calculate the residuals series to check if it is stationary, meaning if the 2 price series move together.
```

```
        var residuals = new double[x.Length];
        for (int i = 0; i < x.Length; i++)
        {
            residuals[i] = y[i] - (intercept + slope * x[i]);
        }

        // Check if the residuals are stationary using the augmented Dickey-Fuller test.
        if (ADFTest(residuals, Convert.ToDouble(_period)))
        {
            // If cointegrated, update the positional sizing ratio and the spread threshold of the trade trigger.
            _coefficients = new[] { Convert.ToDecimal(slope), Convert.ToDecimal(intercept) };
            _residualSD = Convert.ToDecimal(Measures.StandardDeviation(residuals));
        }
        else
        {
            _coefficients = new[] { 0m, 0m };
            _residualSD = 1e10m;             // An arbitrarily large number that the class A price will never reach.
        }
    }

    private static bool ADFTest(double[] series, double N)
    {
        var n = series.Length;
        var lagged = new double[n - 1];
        var differences = new double[n - 1];

        // Fit linear regression for the residual series on unit root: ΔY_t = α + βY_{t-1} + ε_t.
        for (int i = 1; i < n; i++)
        {
            lagged[i - 1] = series[i - 1];
            differences[i - 1] = series[i] - series[i - 1];
        }

        var regressionResult = SimpleRegression.Fit(lagged, differences);
        var alpha = regressionResult.Item1;  // Intercept
        var beta = regressionResult.Item2;   // Coefficient of lagged term

        // Calculate the ADF statistic and check if the null hypothesis is rejected.
        var adfStatistic = beta / Measures.StandardError(differences);

        // Reject the null hypothesis of a unit root is present if test statistic <= critical value
        //This means no unit root exists for the difference series, and the residuals are stationary.
        var critical = -1.941d + -0.2686d / N + -3.365d / N / N + 31.223d / N / N / N;
        return adfStatistic <= critical;
    }
}
```

Example 2: Custom Money Flow Index

The following algorithm implements a custom Money Flow Index indicator, and updates it manually. We estimate the supply-demand balance of SPY and trade using the average money flow direction.

```
C#

public class CombiningIndicatorsAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private CustomMoneyFlowIndex _customMfi;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Request daily SPY data to feed the indicators to generate trade signals and trade.
        _spy = AddEquity("SPY", dataNormalizationMode: DataNormalizationMode.Raw).Symbol;

        // Create a custom money flow index to generate a trade signal.
        _customMfi = new CustomMoneyFlowIndex(20);

        // Warm up for immediate usage of indicators.
        SetWarmUp(20, Resolution.Daily);
    }

    public override void OnData(Slice slice)
    {
        if (!slice.Bars.TryGetValue(_spy, out var bar))
        {
```

```
        {
            return;
        }
        // Update the custom MFI with the updated trade bar to obtain the updated trade signal.
        _customMfi.Update(bar);
        // Don't trade until warm-up is done.
        if (IsWarmingUp)
        {
            return;
        }
        // Buy if the positive money flow is above the negative, which indicates demand is greater than supply,
driving up the price.
        if (_customMfi > 50)
        {
            SetHoldings(_spy, 1);
        }
        // Sell if the positive money flow is below negative, indicating demand is less than supply, driving down the
price.
        else
        {
            SetHoldings(_spy, -1);
        }
    }

    public class CustomMoneyFlowIndex : TradeBarIndicator, IIndicatorWarmUpPeriodProvider
    {
        private decimal _previousTypicalPrice;
        private RollingWindow<decimal> _negativeMoneyFlow;
        private RollingWindow<decimal> _positiveMoneyFlow;
        public override bool IsReady => _positiveMoneyFlow.IsReady;
        public int WarmUpPeriod => _positiveMoneyFlow.Size;

        public CustomMoneyFlowIndex(int period) : base("CustomMFI")
        {
            _negativeMoneyFlow = new(period);
            _positiveMoneyFlow = new(period);
            _previousTypicalPrice = 0m;
        }

        protected override decimal ComputeNextValue(TradeBar input)
        {
            // Estimate the money flow by averaging the price multiplied by volume.
            var typicalPrice = (input.High + input.Low + input.Close) / 3;
            var moneyFlow = typicalPrice * input.Volume;

            // We need to avoid double rounding errors.
            _negativeMoneyFlow.Add(typicalPrice < _previousTypicalPrice ? moneyFlow: 0);
            _positiveMoneyFlow.Add(typicalPrice > _previousTypicalPrice ? moneyFlow: 0);
            _previousTypicalPrice = moneyFlow;

            // Add the period money flow to calculate the aggregated money flow.
            var positiveMoneyFlowSum = _positiveMoneyFlow.Sum();
            var totalMoneyFlow = positiveMoneyFlowSum + _negativeMoneyFlow.Sum();

            // Set the value to be the positive money flow ratio.
            return totalMoneyFlow == 0 ? 100m : 100m * positiveMoneyFlowSum / totalMoneyFlow;
        }

        public override void Reset()
        {
            _previousTypicalPrice = 0m;
            _negativeMoneyFlow.Reset();
            _positiveMoneyFlow.Reset();
            base.Reset();
        }
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms

Indicators

Indicator Universes

Introduction

An indicator universe uses technical indicators to determine the constituents of the universe. Imagine a universe that only contains assets above their 10-day simple moving average. You can incorporate indicators into any of the types of universes in the Universes chapter . To create an indicator universe, define a helper class that contains the indicators and then define a universe that updates the indicators and selects assets.

Define SymbolData Objects

To make it easy to create and update indicators for each security in the universe, move the indicator logic into a class. In the universe definition, you can create an instance of this class for each security in the universe. The indicators you create in this class should be manual indicators so you can ensure they only update during universe selection.

C#

```
private class SelectionData
{
    public readonly ExponentialMovingAverage Fast;
    public readonly ExponentialMovingAverage Slow;

    public SelectionData()
    {
        Fast = new ExponentialMovingAverage(100);
        Slow = new ExponentialMovingAverage(300);
    }

    public decimal ScaledDelta
    {
        get { return (Fast - Slow)/((Fast + Slow)/2m); }
    }

    public bool Update(DateTime time, decimal value)
    {
        return Fast.Update(time, value) && Slow.Update(time, value);
    }
}
```

You need to use a `SymbolData` class instead of assigning the indicators to the `Fundamental` object because you can't create custom properties on `Fundamental` objects like you can with `Security` objects.

Define the Universe

You need to define SymbolData objects before you define the universe that selects securities.

When your universe function receives an object that contains all the possible securities, create a `SymbolData` object for each new security and update the remaining `SymbolData` objects with their daily price or some other data point. For example, the following universe definition selects US Equities that have the greatest difference between two moving averages.

```C#
using System.Collections.Concurrent;
public class EmaCrossUniverseSelectionAlgorithm : QCAlgorithm
{
    // tolerance to prevent bouncing
    const decimal Tolerance = 0.01m;
    private const int Count = 10;
    // use Buffer+Count to leave a little in cash
    private const decimal TargetPercent = 0.1m;
    private SecurityChanges _changes = SecurityChanges.None;
    // holds our fundamental indicators by symbol
    private readonly ConcurrentDictionary<Symbol, SelectionData> _averages = new ConcurrentDictionary<Symbol,
SelectionData>();

    public override void Initialize()
    {
        UniverseSettings.Asynchronous = true;
        UniverseSettings.Leverage = 2.0m;
        UniverseSettings.Resolution = Resolution.Daily;
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        AddUniverse(fundamental =>
        {
            return (from f in fundamental
                    // grab th SelectionData instance for this symbol
                    let avg = _averages.GetOrAdd(f.Symbol, sym => new SelectionData())
                    // Update returns true when the indicators are ready, so don't accept until they are
                    where avg.Update(f.EndTime, f.AdjustedPrice)
                    // only pick symbols who have their 50 day ema over their 100 day ema
                    where avg.Fast > avg.Slow*(1 + Tolerance)
                    // prefer symbols with a larger delta by percentage between the two averages
                    orderby avg.ScaledDelta descending
                    // we only need to return the symbol and return 'Count' symbols
                    select f.Symbol).Take(Count);
        });
    }
}
```

Examples

The following examples demonstrate some common practices for using indicator universes.

Example 1: Universe Selection

The following algorithm selects the stocks within 5% of the 1-year maximum price among the top 100 liquid stocks. To do so, we make use of Maximum indicator to do so. Then, we hold the stocks with price > EMA > SMA , which indicates an upward accelerating trend.

```C#
public class ManualIndicatorAlgorithm : QCAlgorithm
{
    private Dictionary<Symbol, Maximum> _maximumBySymbol = new();
    private Dictionary<Symbol, SymbolData> _symbolData = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Select a popularity-based universe with indicators in a Selection function.
        AddUniverse(Selection);

        // Set a scheduled event to rebalance daily on the daily indicator signals.
        Schedule.On(
            DateRules.EveryDay(),
            TimeRules.At(9, 31),
            Rebalance
        );
    }

    private IEnumerable<Symbol> Selection(IEnumerable<Fundamental> fundamentals)
    {
```

```
        var selected = new List<Symbol>();

        // Initially filtered for the top 100 liquid stocks first.
        var filtered = fundamentals.OrderByDescending(f => f.DollarVolume)
            .Take(100)
            .ToList();

        foreach (var f in filtered)
        {
            if (!_maximumBySymbol.TryGetValue(f.Symbol, out var maximum))
            {
                maximum = new(252);
                // Warm up the Maximum indicator for its readiness to use immediately.
                var history = History<TradeBar>(f.Symbol, 252, Resolution.Daily);
                foreach (var bar in history)
                {
                    maximum.Update(bar.EndTime, bar.Close);
                }
                _maximumBySymbol[f.Symbol] = maximum;
            }
            else
            {
                // Update the indicator with the last known adjusted price daily.
                maximum.Update(f.EndTime, f.AdjustedPrice);
            }

            // Select to trade if the current price is within 5% of the maximum price of the last year.
            // Close to the maximum price provides evidence of high popularity for the fund to support the trend.
            if (f.AdjustedPrice >= maximum * 0.95m)
            {
                selected.Add(f.Symbol);
            }
        }

        return selected;
    }

    private void Rebalance()
    {
        var symbolsToBuy = new List<Symbol>();

        foreach (var (symbol, symbolData) in _symbolData)
        {
            // Buy the stocks whose prices are above the EMA and the EMA is above the SMA, meaning their trend is
upward accelerating.
            if (symbolData.IsReady && Securities[symbol].Price > symbolData.Ema && symbolData.Ema > symbolData.Sma)
            {
                symbolsToBuy.Add(symbol);
            }
        }

        // Equally invest in selected stocks to dissipate capital risk evenly.
        var count = symbolsToBuy.Count;
        if (count > 0)
        {
            var targets = symbolsToBuy.Select(symbol => new PortfolioTarget(symbol, 1m / count)).ToList();
            // Liquidate the positions that are not on an upward trend or are not popular anymore.
            SetHoldings(targets, liquidateExistingHoldings: true);
        }
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach (var added in changes.AddedSecurities)
        {
            // Instantiate a new instance of SymbolData object to hold indicators to use.
            _symbolData[added.Symbol] = new SymbolData(this, added.Symbol);
        }

        foreach (var removed in changes.RemovedSecurities)
        {
            // Remove the data subscription of indicators to release computation resources when leaving the universe.
            if (_symbolData.Remove(removed.Symbol, out var symbolData))
            {
                symbolData.Dispose();
            }
        }
    }

    private class SymbolData
    {
```

```
    {
        private QCAlgorithm _algorithm;
        public Symbol Symbol { get; set; }
        public ExponentialMovingAverage Ema { get; set; }
        public SimpleMovingAverage Sma { get; set; }

        public bool IsReady => Ema.IsReady && Sma.IsReady;

        public SymbolData(QCAlgorithm algorithm, Symbol symbol)
        {
            _algorithm = algorithm;
            Symbol = symbol;

            // Create an EMA and an SMA manual indicator for trade signal generation.
            Ema = new ExponentialMovingAverage(20);
            Sma = new SimpleMovingAverage(20);

            // Warm up the indicators to ensure their readiness to use them immediately.
            algorithm.WarmUpIndicator(symbol, Ema, Resolution.Daily);
            algorithm.WarmUpIndicator(symbol, Sma, Resolution.Daily);

            // Subscribe to the indicators to update daily price data for updated trade signal generation.
            algorithm.RegisterIndicator(symbol, Ema, Resolution.Daily);
            algorithm.RegisterIndicator(symbol, Sma, Resolution.Daily);
        }

        public void Dispose()
        {
            Ema.Reset();
            Sma.Reset();
            // Cancel data subscription to release computation resources.
            _algorithm.DeregisterIndicator(Ema);
            _algorithm.DeregisterIndicator(Sma);
        }
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
EmaCrossUniverseSelectionAlgorithm.cs C#

Indicators

Rolling Window

Introduction

A `RollingWindow` is an array of a fixed-size that holds trailing data. It's more efficient to use `RollingWindow` objects to hold periods of data than to make multiple historical data requests . With a `RollingWindow` , you just update the latest data point while a `History` call fetches all of the data over the period you request. `RollingWindow` objects operate on a first-in, first-out process to allow for reverse list access semantics with positive and negative indexing. Index 0 refers to the most recent item in the window and the largest index refers to the last item in the window. Index -1 refers to the latest item as well.

Create Rolling Windows

To create rolling windows, instantiate the `RollingWindow` object of a given type with a maximum size.

```
C#

closePriceWindow = new RollingWindow<decimal>(4);
tradeBarWindow = new RollingWindow<TradeBar>(2);
quoteBarWindow = new RollingWindow<QuoteBar>(2);
```

Add Data

To add data to a `RollingWindow` , call the `Add` method.

```
C#

closePriceWindow.Add(data["SPY"].Close);
tradeBarWindow.Add(data["SPY"]);
quoteBarWindow.Add(data["EURUSD"]);
```

To update the data at a specific index, set the value for that index. If the index doesn't currently exist, it increases the size and fills the empty indices with a default value (zero or `null`).

```
C#

closePriceWindow[0] = data["SPY"].Close;
tradeBarWindow[0] = data["SPY"];
quoteBarWindow[0] = data["EURUSD"];
```

Warm Up

To warm up a `RollingWindow` , make a history request and then iterate through the result to add the data to the `RollingWindow` .

```C#
public class WarmUpRollingWindowAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        var spy = AddEquity("SPY", Resolution.Daily).Symbol;
        var historyTradeBar = History<TradeBar>(spy, 10, Resolution.Daily);
        var historyQuoteBar = History<QuoteBar>(spy, 10, Resolution.Minute);

        // Warm up the close price and trade bar rolling windows with the previous 10-day trade bar data
        var closePriceWindow = new RollingWindow<decimal>(10);
        var tradeBarWindow = new RollingWindow<TradeBar>(10);
        foreach (var tradeBar in historyTradeBar)
        {
            closePriceWindow.Add(tradeBar.Close);
            tradeBarWindow.Add(tradeBar);
        }

        // Warm up the quote bar rolling window with the previous 10-minute quote bar data
        var quoteBarWindow = new RollingWindow<QuoteBar>(10);
        foreach (var quoteBar in historyQuoteBar)
        {
            quoteBarWindow.Add(quoteBar);
        }
    }
}
```

Check Readiness

To check if a `RollingWindow` is full, use its `IsReady` flag.

```C#
if (!closePriceWindow.IsReady)
{
    return;
}
```

Adjust Size

To adjust the `RollingWindow` size, set the `Size` property.

```C#
closePriceWindow.Size = 3;
tradeBarWindow.Size = 3;
quoteBarWindow.Size = 3;
```

When you decrease the size, it removes the oldest values that no longer fit in the `RollingWindow` . When you explicitly increase the `Size` member, it doesn't automatically add any new elements to the `RollingWindow` . However, if you set the value of an index in the `RollingWindow` and the index doesn't currently exist, it fills the empty indices with a default value (zero or `null`). For example, the following code increases the `Size` to 10, sets the 10th element to 3, and sets the 4th-9th elements to the default value:

```C#
closePriceWindow[9] = 3;
```

Access Data

`RollingWindow` objects operate on a first-in, first-out process to allow for reverse list access semantics with positive and negative indexing. Index 0 refers to the most recent item in the window and the largest index refers to the last item in the window. Index -1 refers to the latest item as well.

```
var currentClose = closePriceWindow[0];
var previousClose = closePriceWindow[1];
var oldestClose = closePriceWindow[closePriceWindow.Count-1];
oldestClose = closePriceWindow[-1];   // Negative indexing
```

To get the item that was most recently removed from the `RollingWindow` , use the `MostRecentlyRemoved` property.

```
var removedClose = closePriceWindow.MostRecentlyRemoved;
```

Combine with Indicators

The `Window` property of the indicators is a built-in `RollingWindow` that stores historical values. It holds a collection of `IndicatorDataPoint` objects, enabling quick access to the most recent historical indicator values for analysis, calculations, or comparisons in trading and financial strategies. Its default size 2, but you can adjust its size. To warm up the indicator and its `RollingWindow` , call the `IndicatorHistory` method with the window size as an argument.

```
public class IndicatorRollingWindowAlgorithm : QCAlgorithm
{
    // Manual SMA indicator example.
    private SimpleMovingAverage _sma = new(20);
    private Beta _beta;

    public override void Initialize()
    {
        var spy = AddEquity("SPY").Symbol;
        var qqq = AddEquity("QQQ").Symbol;

        // Automatic Beta indicator example.
        _beta = B(spy, qqq, 20);

        // Adjust the window size to 5 to access the previous 5 indicator data points.
        _sma.Window.Size = 5;
        _beta.Window.Size = 5;

        // Warm up the indicator and window for immediate usage.
        IndicatorHistory(_sma, spy, _sma.Window.Size, Resolution.Daily);
        // For indicators using 2+ symbols, use an iterable symbol list.
        IndicatorHistory(_beta, new[] { spy, qqq },  _beta.Window.Size, Resolution.Daily);
    }
}
```

Combine with Consolidators

To store a history of consolidated bars , in the consolidation handler, add the consolidated bar to the `RollingWindow` .

```
_consolidator.DataConsolidated += (sender, consolidatedBar) => tradeBarWindow.Add(consolidatedBar);
```

Market Session

The `session` property of a `Security` object contains the open, high, low, close, volume, and open interest of trading sessions. The `Session` object is a `RollingWindow` object with size of zero by default. To enable market session tracking, set the size of the `Session` object to 2, so the previous session data is at index 1.

```
var session = Securities[_symbol].Session;
session.Size = 2;
var todayOpen = session.Open;
var todayHigh = session.High;
var todayLow = session.Low;
var todayCumulativeVolume = session.Volume;
if (session.IsReady)
{
    var yesterdayClose = session[1].Close;
}
```

Increase the size of the `Session` object to store more session data. You can use cached data in the `session` property instead of requesting historical daily data .

```
var equity = AddEquity("SPY");
// Increase the session lookback to 252 from default (0)
equity.Session.Size = 252;
// Get all closing prices in the session history
var closes = session.OrderBy(x => x.EndTime).Select(x => x.Close);
```

Cast to Other Types

You can cast a `RollingWindow` to a list. If you cast it to a list, reverse the list so the most recent element is at the last index of the list. This is the order the elements would be in if you added the elements to the list with the `Add` method.

```
var closes = closePriceWindow.Reverse().ToList();
```

Delete Data

To remove all of the elements from a `RollingWindow` , call the `Reset` method.

```
closePriceWindow.Reset();
```

Examples

The following examples demonstrate some common practices for rolling windows.

Example 1: Price Actions

The following algorithm saves the trailing 3 `TradeBar` objects into a `RollingWindow` . When it identifies a volume contraction breakout price action pattern on the SPY, it buys to ride on the capital inflow. To exit positions, it places a 2% take profit and 1% stop loss order in the `OnOrderEvent` method.

```
public class RollingWindowAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    // Set up a rolling window to hold the last 3 trade bars for price action detection as the trade signal.
    private RollingWindow<TradeBar> _windows = new(3);

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Add SPY data for signal generation and trading.
        _spy = AddEquity("SPY", Resolution.Minute).Symbol;
```

```
        // Warm up the rolling window.
        var history = History<TradeBar>(_spy, 3, Resolution.Minute);
        foreach (var bar in history)
        {
            _windows.Add(bar);
        }
    }

    public override void OnData(Slice slice)
    {
        if (slice.Bars.TryGetValue(_spy, out var bar))
        {
            // Trade the price action if the previous bars fulfill a contraction breakout.
            if (ContractionAction() && BreakoutAction(bar.Close))
            {
                SetHoldings(_spy, 0.5m);
            }

            // Add the current bar to the window.
            _windows.Add(bar);
        }
    }

    private bool ContractionAction()
    {
        // We trade contraction type price action, where the buying preesure is increasing.
        // 1. The last 3 bars are green.
        // 2. The price is increasing in trend.
        // 3. The trading Volume is increasing as well.
        // 4. The range of the bars are decreasing.
        return _windows[2].Close > _windows[2].Open &&
            _windows[1].Close > _windows[1].Open &&
            _windows[0].Close > _windows[0].Open &&
            _windows[0].Close > _windows[1].Close && _windows[1].Close > _windows[2].Close &&
            _windows[0].Volume > _windows[1].Volume && _windows[1].Volume > _windows[2].Volume &&
            _windows[2].Close - _windows[2].Open > _windows[1].Close - _windows[1].Open &&
            _windows[1].Close - _windows[1].Open > _windows[0].Close - _windows[0].Open;
    }

    private bool BreakoutAction(decimal currentPrice)
    {
        // Trade breakout from contraction: the breakout should be much greater than the contracted range of the last
bar.
        return currentPrice - _windows[0].Close > (_windows[0].Close - _windows[0].Open) * 2m;
    }

    public override void OnOrderEvent(OrderEvent orderEvent)
    {
        if (orderEvent.Status == OrderStatus.Filled)
        {
            if (orderEvent.Ticket.OrderType == OrderType.Market)
            {
                // Stop loss order at 1%.
                var stopPrice = orderEvent.FillQuantity > 0m ? orderEvent.FillPrice * 0.99m : orderEvent.FillPrice *
1.01m;
                StopMarketOrder(_spy, -Portfolio[_spy].Quantity, stopPrice);
                // Take profit order at 2%.
                var takeProfitPrice = orderEvent.FillQuantity > 0m ? orderEvent.FillPrice * 1.02m :
orderEvent.FillPrice * 0.98m;
                LimitOrder(_spy, -Portfolio[_spy].Quantity, takeProfitPrice);
            }
            else if (orderEvent.Ticket.OrderType == OrderType.StopMarket || orderEvent.Ticket.OrderType ==
OrderType.Limit)
            {
                // Cancel open orders if the stop loss or take profit order fills.
                Transactions.CancelOpenOrders();
            }
        }
    }
}
```

Example 2: Bid-Ask Spread

The following algorithm trades the microeconomy of SPY's supply-demand relationship. It buys when the current bid-ask spread is less than average spread over the last 20 QuoteBar objects, indicating demand is approaching supply. When the

spread is greater than the average, it shorts SPY. To save the last spread values, it uses a `RollingWindow` .

```C#
public class RollingWindowAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    // Set up a rolling window to hold the last 20 bar's bid-ask spread for trade signal generation.
    private RollingWindow<decimal> _windows = new(20);

    public override void Initialize()
    {
        SetStartDate(2024, 12, 1);
        SetEndDate(2024, 12, 31);

        // Add SPY data for signal generation and trading.
        _spy = AddEquity("SPY", Resolution.Minute).Symbol;

        // Warm up for the rolling window with quote data.
        var history = History<QuoteBar>(_spy, 20, Resolution.Minute);
        foreach (var bar in history)
        {
            _windows.Add(bar.Ask.Close - bar.Bid.Close);
        }
    }

    public override void OnData(Slice slice)
    {
        if (slice.QuoteBars.TryGetValue(_spy, out var bar))
        {
            // Update the window with the current bid-ask spread.
            var spread = bar.Ask.Close - bar.Bid.Close;
            _windows.Add(spread);

            // Buy if the current spread is smaller than average, indicating demand is approaching supply. Buy force
will drive up price.
            if (spread < _windows.Average())
            {
                SetHoldings(_spy, -0.5m);
            }
            // Short if the current spread is larger than average, indicating supply is gradually overwhelming
demand.
            else if (spread > _windows.Average())
            {
                SetHoldings(_spy, 0.5m);
            }
        }
    }
}
```

Example 3: EMA Acceleration

The following algorithm trades SPY based on the direction and acceleration of its exponential moving average (EMA). It buys when the EMA is increasing and accelerating upward, indicating a strong uptrend. It short sells when the EMA is decreasing and accelerating downward, indicating a strong downtrend. To compare the current EMA value to its previous values, the algorithm utilizes the EMA's internal `RollingWindow` .

```C#
public class RollingWindowAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private ExponentialMovingAverage _ema;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Add SPY data for trading.
        _spy = AddEquity("SPY", Resolution.Minute).Symbol;

        // Set up an automatic EMA indicator for trade signal generation.
        _ema = EMA(_spy, 20, Resolution.Daily);
        // Extend the size of the EMA's internal rolling window so we can access its historical values.
        _ema.Window.Size = 3;
        // Warm-up the indicator and its internal rolling window
        IndicatorHistory(_ema, _spy, _ema.Window.Size);

        // Schedule an event to rebalance SPY position at daily market open.
        Schedule.On(DateRules.EveryDay(_spy), TimeRules.AfterMarketOpen(_spy, 0), Rebalance);
    }

    private void Rebalance()
    {
        // Buy if the current EMA is increasing with acceleration, indicating a strong uptrend.
        if (_ema.Window[1] < _ema && _ema.Window[0] - _ema.Window[1] > _ema.Window[1] - _ema.Window[2])
        {
            SetHoldings(_spy, 0.5m);
        }
        // Short if the current EMA is decreasing with acceleration, indicating a strong downtrend.
        else if (_ema.Window[1] > _ema && _ema.Window[0] - _ema.Window[1] < _ema.Window[1] - _ema.Window[2])
        {
            SetHoldings(_spy, -0.5m);
        }
        // Liquidate if no strong trend indicated.
        else if (Portfolio.Invested)
        {
            Liquidate(_spy);
        }
    }
}
```

Example 4: Market Session as History

Using `Session` instead of history requests or rolling window to cache the last 252 OHLCV bars, the following algorithm buys SPY accordingly when the market opens above the 252-day max of closing prices.

C#

```
public class MarketSessionAsDailyHistory : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        var equity = AddEquity("SPY");

        // Increase the session lookback to 252 from default (0)
        // since our strategy break out the 252-day max of closing prices
        equity.Session.Size = 252;
        // Warm up with daily data for speed
        SetWarmUp(equity.Session.Size, Resolution.Daily);

        Schedule.On(DateRules.EveryDay("SPY"), TimeRules.AfterMarketOpen("SPY", 1), () =>
        {
            // Break-out strategy. If the market opens above the 252-day max of closing prices,
            // allocate all in SPY. Liquidate if it opens below the mean.

            var session = Securities["SPY"].Session;
            // The session is ready when the number of samples is greater than the size (252)
            if (IsWarmingUp || !session.IsReady)
                return;

            var closes = session.OrderBy(x => x.EndTime).Select(x => x.Close);
            var open = session.Open;
            var max = closes.Max();
            var mean = closes.Average();

            // Invest all in SPY if the open break out the 252-day max and liquidate below the mean
            if (open > max)
                SetHoldings("SPY", 1, tag: $"_open={open:F2}, _max={max:F2}");
            if (open < mean)
                Liquidate(symbol: "SPY", tag:$"_open={open:F2}, _mean={mean:F2}");
        });
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm
RollingWindowAlgorithm.cs C# MultipleSymbolConsolidationAlgorithm.cs C#

Object Store

Introduction

The Object Store is a file system that you can use in your algorithms to save, read, and delete data. The Object Store is organization-specific, so you can save or read data from the same Object Store in all of your organization's projects. The Object Store works like a key-value storage system where you can store regular strings, JSON encoded strings, XML encoded strings, and bytes. You can access the data you store in the Object Store from backtests, the Research Environment, and live algorithms.

Get All Stored Data

To get all of the keys and values in the Object Store, iterate through the `ObjectStore` property.

```
C#
foreach (var kvp in ObjectStore)
{
    var key = kvp.Key;
    var value = kvp.Value;
}
```

To iterate through just the keys in the Object Store, iterate through the `Keys` property.

```
C#
foreach (var key in ObjectStore.Keys)
{
    continue;
}
```

Create Sample Data

You need some data to store data in the Object Store.

Follow these steps to create some sample data:

1. Create a dictionary.

   ```
   C#
   var dictSample = new Dictionary<string, int> { {"One", 1}, {"Two", 2}, {"Three", 3} };
   ```

2. Create a `string` .

   ```
   C#
   var stringSample = "My string";
   ```

3. Create a `Bytes` object.

   ```
   C#
   var bytesSample = Encoding.UTF8.GetBytes("My String");
   ```

4. Convert the dictionary to an `XML` -formatted object.

```
var xmlSample = new XElement("sample",
    dictSample.Select(kvp => new XElement(kvp.Key, kvp.Value)));
Log(xmlSample.ToString());
```

```
<sample>
  <One>1</One>
  <Two>2</Two>
  <Three>3</Three>
</sample>
```

Save Data

The Object Store saves objects under a key-value system. If you save objects in backtests, you can access them from the Research Environment. To avoid slowing down your backtests, save data once in the `OnEndOfAlgorithm` event handler. In live trading, you can save data more frequently like at the end of a `Train` method or after universe selection.

If you run algorithms in QuantConnect Cloud, you need storage create permissions to save data in the Object Store.

If you don't have data to store, create some sample data .

You can save the following types of objects in the Object Store:

- `Bytes` objects
- `string` objects
- JSON objects
- XML-formatted objects

To store data, you need to provide a key. If you provide a key that is already in the Object Store, it will overwrite the data at that location. To avoid overwriting objects from other projects in your organization, prefix the key with your project ID. You can find the project ID in the URL of your browser when you open a project. For example, the ID of the project at quantconnect.com/project/12345 is 12345.

Strings

To save a `string` object, call the `Save` or `SaveString` method.

```
var saveSuccessful = ObjectStore.Save($"{ProjectId}/stringKey", stringSample);
```

JSON

To save a JSON object, call the `SaveJson<T>` method. This method helps to serialize the data into JSON format.

```
var saveSuccessful = ObjectStore.SaveJson<Dictionary<string, int>>($"{ProjectId}/jsonKey", dictSample);
```

XML

To save an XML-formatted object, call the `SaveXml<T>` method.

```
var saveSuccessful = ObjectStore.SaveXml<XElement>($"{ProjectId}/xmlKey", xmlSample);
```

Bytes

To save a `Bytes` object (for example, zipped data), call the `SaveBytes` method.

```C#
var saveSuccessful = ObjectStore.SaveBytes($"{ProjectId}/bytesKey", bytesSample)

var zippedDataSample = Compression.ZipBytes(Encoding.UTF8.GetBytes(stringSample), "data");
var saveSuccessful = ObjectStore.SaveBytes($"{ProjectId}/bytesKey.zip", zippedDataSample);
```

Read Data

Read data from the Object Store to import algorithm variables between deployments, import data from the Research Environment, or load trained machine learning models. To read data from the Object Store, you need to provide the key you used to store the object.

You can load the following types of objects from the Object Store:

- `Bytes` objects
- `string` objects
- JSON objects
- XML-formatted objects

Before you read data from the Object Store, check if the key exists.

```C#
if (ObjectStore.ContainsKey(key))
{
    // Read data
}
```

Strings

To read a `string` object, call the `Read` or `ReadString` method.

```C#
var stringData = ObjectStore.Read($"{ProjectId}/stringKey");
```

JSON

To read a JSON object, call the `ReadJson<T>` method.

```C#
var jsonData = ObjectStore.ReadJson<Dictionary<string, int>>($"{ProjectId}/jsonKey");
```

XML

To read an XML-formatted object, call the `ReadXml<T>` method.

```C#
var xmlData = ObjectStore.ReadXml<XElement>($"{ProjectId}/xmlKey");
```

If you created the XML object from a dictionary, reconstruct the dictionary.

```C#
var dict = xmlData.Elements().ToDictionary(x => x.Name.LocalName, x => int.Parse(x.Value));
```

Bytes

To read a `Bytes` object, call the `ReadBytes` method.

```C#
var bytesData = ObjectStore.ReadBytes($"{ProjectId}/bytesKey");
```

Delete Data

Delete objects in the Object Store to remove objects that you no longer need. If you run algorithms in QuantConnect Cloud, you need storage delete permissions to delete data from the Object Store.

To delete objects from the Object Store, call the `Delete` method. Before you delete data, check if the key exists. If you try to delete an object with a key that doesn't exist in the Object Store, the method raises an exception.

```C#
if (ObjectStore.ContainsKey(key))
{
    ObjectStore.Delete(key);
}
```

To delete all of the content in the Object Store, iterate through all the stored data.

```C#
foreach (var kvp in ObjectStore)
{
    ObjectStore.Delete(kvp.Key);
}
```

Cache Data

When you write to or read from the Object Store, the algorithm caches the data. The cache speeds up the algorithm execution because if you try to read the Object Store data again with the same key, it returns the cached data instead of downloading the data again. The cache speeds up execution, but it can cause problems if you are trying to share data between two nodes under the same Object Store key. For example, consider the following scenario:

1. You open project A and save data under the key `123` .
2. You open project B and save new data under the same key `123` .
3. In project A, you read the Object Store data under the key `123` , expecting the data from project B, but you get the original data you saved in step #1 instead.

   You get the data from step 1 instead of step 2 because the cache contains the data from step 1.

To clear the cache, call the `Clear` method.

```C#
ObjectStore.Clear();
```

Get File Path

To get the file path for a specific key in the Object Store, call the `GetFilePath` method. If the key you pass to the method doesn't already exist in the Object Store, it's added to the Object Store.

```
C#
var filePath = ObjectStore.GetFilePath(key);
```

Storage Quotas

If you run algorithms locally, you can store as much data as your hardware will allow. If you run algorithms in QuantConnect Cloud, you must stay within your storage quota . To find your storage quota programmatically, use the `MaxSize` property for storage size and `MaxFiles` property for the maximum number of files.

```
C#
var maxSize = ObjectStore.MaxSize;
var maxFiles = ObjectStore.MaxFiles;
```

If you need more storage space, edit your storage plan .

Example for DataFrames

Follow these steps to create a DataFrame, save it into the Object Store, and load it from the Object Store:

1. Get some historical data.

   ```
   C#
   var spy = AddEquity("SPY").Symbol;
   var history = History(Securities.Keys, 360, Resolution.Daily);
   ```

2. Create a DataFrame.

   ```
   C#
   using Microsoft.Data.Analysis; //

   var columns = new DataFrameColumn[] {
       new DateTimeDataFrameColumn("Time", history.Select(x => (DateTime)x[spy].EndTime)),
       new DecimalDataFrameColumn("SPY Open", history.Select(x => (decimal)x[spy].Open)),
       new DecimalDataFrameColumn("SPY High", history.Select(x => (decimal)x[spy].High)),
       new DecimalDataFrameColumn("SPY Low", history.Select(x => (decimal)x[spy].Low)),
       new DecimalDataFrameColumn("SPY Close", history.Select(x => (decimal)x[spy].Close))
   };
   var df = new DataFrame(columns);
   ```

3. Get the file path for a specific key in the Object Store.

   ```
   C#
   var filePath = ObjectStore.GetFilePath("df_to_csv");
   ```

4. Call the SaveCsv method to save the DataFrame in the Object Store as a CSV file.

   ```
   C#
   DataFrame.SaveCsv(df, filePath);    // File size: 26520 bytes
   ```

5. Call the LoadCsv method to load the CSV file from the Object Store.

```
C#

var reread = DataFrame.LoadCsv(filePath);
```

Example for Plotting

You can use the Object Store to plot data from your backtests and live algorithm in the Research Environment. The following example demonstrates how to plot a Simple Moving Average indicator that's generated during a backtest.

1. Create a algorithm, add a data subscription, and add a simple moving average indicator.

```
C#

public class ObjectStoreChartingAlgorithm : QCAlgorithm
{
    private SimpleMovingAverage _sma;
    private string _content;

    public override void Initialize()
    {
        AddEquity("SPY", Resolution.Minute);
        _sma = SMA("SPY", 22);
    }
}
```

   The algorithm will save `_content` to the Object Store.

2. Save the indicator data as `string` in `_content` .

```
C#

public override void OnData(Slice data)
{
    _content += $"{_sma.Current.EndTime},{_sma}\n";
}
```

3. In the OnEndOfAlgorithm method, save the indicator data to the Object Store.

```
C#

public override void OnEndOfAlgorithm()
{
    ObjectStore.Save("sma_values_csharp", _content);
}
```

4. Open the Research Environment and create a `QuantBook` .

```
C#

// Execute the following command in first
#load "../Initialize.csx"

// Create a QuantBook object
#load "../QuantConnect.csx"
using QuantConnect;
using QuantConnect.Research;

var qb = new QuantBook();
```

5. Read the indicator data from the Object Store.

```
C#

var content = qb.ObjectStore.Read("sma_values_csharp");
```

   The key you provide must be the same key you used to save the object.

6. Import the `Plotly.NET` and `Plotly.NET.LayoutObjects` packages.

```C#
#r "../Plotly.NET.dll"
using Plotly.NET;
using Plotly.NET.LayoutObjects;
```

7. Create the `Layout` object and set the `title` , `xaxis` , and `yaxis` properties.

```C#
var layout = new Layout();
layout.SetValue("title", Title.init("SMA"));

var xAxis = new LinearAxis();
xAxis.SetValue("title", "Time");
layout.SetValue("xaxis", xAxis);

var yAxis = new LinearAxis();
yAxis.SetValue("title", "SMA");
layout.SetValue("yaxis", yAxis);
```

8. Convert the data to a list of `DateTime` objects for the chart x-axis and a list of `decimal` objects for the chart y-axis, then create a `Chart2D.Chart.Line` object with the data.

```C#
var index = new List<DateTimee>();
var values = new List<decimal>();

foreach (var line in content.Split('\n'))
{
    var csv = line.Split(',');
    if (csv.Length > 1)
    {
        index.Add(Parse.DateTime(csv[0]));
        values.Add(decimal.Parse(csv[1]));
    }
}

var chart = Chart2D.Chart.Linee<DateTime, decimal, stringe>(index, values);
```

9. Apply the layout to the `Line` object and create the `HTML` object.

```C#
chart.WithLayout(layout);
var result = HTML(GenericChart.toChartHTML(chart));
```

```
C#
public class ObjectStoreChartingAlgorithm : QCAlgorithm
{
    private SimpleMovingAverage _sma;
    private string _content;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        AddEquity("SPY", Resolution.Minute);
        // Create SMA indicator for referencing.
        _sma = SMA("SPY", 22);
    }

    public override void OnData(Slice data)
    {
        // Cache the indicator data point to save it.
        _content += $"{_sma.Current.EndTime},{_sma}\n";
    }

    public override void OnEndOfAlgorithm()
    {
        // Save the indicator values to object store for logging.
        ObjectStore.Save("sma_values_csharp", _content);
    }
}
```

Example of Custom Data

Follow these steps to use the Object Store as a data source for custom data :

1. Create a custom data class that defines a storage key and implements the `GetSource` method.

```
C#
public class Bitstamp : TradeBar
{
    public static string KEY = "bitstampusd.csv";

    public override SubscriptionDataSource GetSource(SubscriptionDataConfig config, DateTime date, bool
isLiveMode)
    {
        return new SubscriptionDataSource(KEY, SubscriptionTransportMedium.ObjectStore);
    }
}
```

2. Create an algorithm that downloads data from an external source and saves it to the Object Store .

```
C#
public class ObjectStoreCustomDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        if (!ObjectStore.ContainsKey(Bitstamp.KEY))
        {
            var url =
"https://raw.githubusercontent.com/QuantConnect/Documentation/master/Resources/datasets/custom-
data/bitstampusd.csv";
            var content = Download(url);
            ObjectStore.Save(Bitstamp.KEY, content);
        }
    }
}
```

3. Call the `AddData` method to subscribe to the custom type .

```C#
public class ObjectStoreCustomDataAlgorithm : QCAlgorithm
{
    private Symbol _customDataSymbol;
    public override void Initialize()
    {
        _customDataSymbol = AddData<Bitstamp>("BTC").Symbol;
    }
}
```

4. Implement the `Reader` method for the custom data class.

```C#
public class Bitstamp : TradeBar
{
    public override BaseData Reader(SubscriptionDataConfig config, string line, DateTime date, bool isLiveMode)
    {
        //Example Line Format:
        //Date      Open   High    Low     Close   Volume (BTC)    Volume (Currency)   Weighted Price
        //2011-09-13 5.8    6.0     5.65    5.97    58.37138238,    346.0973893944      5.929230648356
        if (!char.IsDigit(line.Trim()[0]))
        {
            return null;
        }

        var coin = new Bitstamp() {Symbol = config.Symbol};

        var data = line.Split(',');
        coin.Value = data[4].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
        if (coin.Value == 0)
        {
            return null;
        }

        coin.Time = DateTime.Parse(data[0], CultureInfo.InvariantCulture);
        coin.EndTime = coin.Time.AddDays(1);
        coin.Open = data[1].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
        coin.High = data[2].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
        coin.Low = data[3].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
        coin.VolumeBTC = data[5].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
        coin.Volume = data[6].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
        coin.WeightedPrice = data[7].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
        coin.Close = coin.Value;
        return coin;
    }
}
```

5. To confirm your algorithm is receiving the custom data, request some historical data , then log and plot the data from the `Slice` object.

```C#
public class ObjectStoreCustomDataAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        var history = History(_customDataSymbol, 200, Resolution.Daily);
        Debug($"We got {history.Count()} items from historical data request of {_customDataSymbol}.");
    }
    public void OnData(Bitstamp data)
    {
        Log($"{data.EndTime}: Close: {data.Close}");
        Plot(_customDataSymbol, "Price", data.Close);
    }
}
```

The following algorithm provides a full example of a sourcing custom data from the Object Store:

C#

```
public class CustomDataBitstampAlgorithm : QCAlgorithm
{
    private Symbol _customDataSymbol;

    public override void Initialize()
    {
        SetStartDate(2012, 9, 13);
        SetEndDate(2021, 6, 20);

        // Load the custom data from ObjectStore.
        if (!ObjectStore.ContainsKey(Bitstamp.KEY))
        {
            var url = "https://raw.githubusercontent.com/QuantConnect/Documentation/master/Resources/datasets/custom-
data/bitstampusd.csv";
            var content = Download(url);
            ObjectStore.Save(Bitstamp.KEY, content);
        }

        // Request custom data for plotting.
        // Define the symbol and "type" of our generic data.
        _customDataSymbol = AddData<Bitstamp>("BTC", Resolution.Daily).Symbol;

        // Example historical call.
        var history = History(_customDataSymbol, 200, Resolution.Daily);
        Debug($"We got {history.Count()} items from historical data request of {_customDataSymbol}.");
    }

    public void OnData(Bitstamp data)
    {
        // Plot on updated data.
        Log($"{data.EndTime}: Close: {data.Close}");
        Plot(_customDataSymbol, "Price", data.Close);
    }

    public class Bitstamp : TradeBar
    {
        public static string KEY = "bitstampusd.csv";

        public override SubscriptionDataSource GetSource(SubscriptionDataConfig config, DateTime date, bool
isLiveMode)
        {
            // Load data from object store.
            return new SubscriptionDataSource(KEY, SubscriptionTransportMedium.ObjectStore);
        }

        public decimal WeightedPrice = 0;
        public decimal VolumeBTC = 0;

        public override BaseData Reader(SubscriptionDataConfig config, string line, DateTime date, bool isLiveMode)
        {
            //Example Line Format:
            //Date      Open   High    Low     Close   Volume (BTC)    Volume (Currency)   Weighted Price
            //2011-09-13 5.8    6.0     5.65    5.97    58.37138238,    346.0973893944      5.929230648356
            if (!char.IsDigit(line.Trim()[0]))
            {
                return null;
            }

            var coin = new Bitstamp() {Symbol = config.Symbol};

            var data = line.Split(',');
            coin.Value = data[4].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
            if (coin.Value == 0)
            {
                return null;
            }

            coin.Time = DateTime.Parse(data[0], CultureInfo.InvariantCulture);
            coin.EndTime = coin.Time.AddDays(1);
            coin.Open = data[1].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
            coin.High = data[2].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
            coin.Low = data[3].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
            coin.VolumeBTC = data[5].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any
```

```
            coin.VolumeBTC = data[5].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
            coin.Volume = data[6].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
            coin.WeightedPrice = data[7].IfNotNullOrEmpty(s => decimal.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture));
            coin.Close = coin.Value;
            return coin;
        }
    }
}
```

Preserve Insights Between Deployments

Follow these steps to use the Object Store to preserve the algorithm state across live deployments:

1. Create an algorithm that defines a storage key and adds insights to the Insight Manager .

```
C#
public class ObjectStoreChartingAlgorithm : QCAlgorithm
{
    private string _insightKey;
    public override void Initialize()
    {
        _insightKey = $"{ProjectId}/insights";
        SetUniverseSelection(new ManualUniverseSelectionModel(QuantConnect.Symbol.Create("SPY",
SecurityType.Equity, Market.USA)));
        SetAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(5), 0.025,
null));
    }
}
```

2. In the OnEndOfAlgorithm event handler of the algorithm, get the Insight objects and save them in the Object Store as a JSON object.

```
C#
public override void OnEndOfAlgorithm()
{
    var insights = Insights.GetInsights(x => x.IsActive(UtcTime));
    ObjectStore.SaveJson(_insightKey, insights);
}
```

3. At the bottom of the `Initialize` method, read the Insight objects from the Object Store and add them to the Insight Manager .

```
C#
if (ObjectStore.ContainsKey(_insightKey))
{
    var insights = ObjectStore.ReadJson<List<Insight>>(_insightKey);
    Insights.AddRange(insights);
}
```

The following algorithm provides a full example of preserving the Insight state between deployments:

```C#
public class ObjectStoreInsightsAlgorithm : QCAlgorithm
{
    private string _insightsKey = "insights";

    public override void Initialize()
    {
        UniverseSettings.Resolution = Resolution.Daily;

        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        _insightsKey = $"{ProjectId}/{_insightsKey}";

        // Read the file with the insights
        if (ObjectStore.ContainsKey(_insightsKey))
        {
            // Load cached JSON key value pair from object store, then add to algorithm insights.
            var insights = ObjectStore.ReadJson<List<Insight>>(_insightsKey);
            Log($"Read {insights.Count} insight(s) from the Object Store");
            Insights.AddRange(insights);

            // Delete the key to reuse it
            ObjectStore.Delete(_insightsKey);
        }

        SetUniverseSelection(new ManualUniverseSelectionModel(QuantConnect.Symbol.Create("SPY", SecurityType.Equity,
Market.USA)));
        SetAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(5), 0.025, null));
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel(Resolution.Daily));
    }

    public override void OnEndOfAlgorithm()
    {
        // Get all active insights.
        var insights = Insights.GetInsights(x => x.IsActive(UtcTime));
        // If we want to save all insights (expired and active), we can use.
        // var insights = Insights.GetInsights(x => true);
        Log($"Save {insights.Count} insight(s) to the Object Store.");
        ObjectStore.SaveJson(_insightsKey, insights);
    }
}
```

Live Trading Considerations

If you update the Object Store from outside of your live trading node and then try to read the new content from inside your live trading algorithm, you may not get the new content because the Object Store caches data to speed up execution. To ensure you get the latest value for a specific key in the Object Store, clear the cache and then read the data .

```C#
ObjectStore.Clear();
if (ObjectStore.ContainsKey(key))
{
    var value = ObjectStore.Read(key);
}
```

Optimization

Optimization

Parameters

Introduction

Parameters are project variables that your algorithm uses to define the value of internal variables like indicator arguments or the length of lookback windows. Parameters are stored outside of your algorithm code, but we inject the values of the parameters into your algorithm when you run a backtest, deploy a live algorithm, or launch an optimization job. To use parameters, set some parameters in your project and then load them into your algorithm.

Set Parameters

The process to set parameter values depends on the environment you use to write algorithms. See the tutorial in one of the following environments:

- Cloud Platform
- Local Platform
- LEAN CLI

Get Parameters

To get a parameter value into your algorithm, call the `GetParameter` method of the algorithm class.

```
C#
// Get the parameter string-value with this matching key.
// Project parameters are useful for optimization or securing variables outside the source-code.
var parameterValue = GetParameter("Parameter Name");
```

The `GetParameter` method returns a string by default. If you provide a default parameter value, the method returns the parameter value as the same data type as the default value. If there are no parameters in your project that match the name you pass to the method and you provide a default value to the method, it returns the default value. The following table describes the arguments the `GetParameter` method accepts:

Argument	Data Type	Description	Default Value
`name`	`string`	The name of the parameter to get	
`defaultValue`	`string/int/double/decimal`	The default value to return	`null`

The following example algorithm gets the values of parameters of each data type:

```C#
// Example to get the values of parameters of each data type:
public class ParameterizedAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // Get the parameter value and return an integer.
        var intParameterValue = GetParameter("intParameterName", 100);

        // Get the parameter value and return a double.
        var doubleParameterValue = GetParameter("doubleParameterName", 0.95);

        // Get the parameter value and return a decimal.
        var decimalParameterValue = GetParameter("decimalParameterName", 0.05m);

        // Get the parameter value as a string.
        var stringParameterValue = GetParameter("parameterName", "defaultStringValue");

        // Cast the string to an integer.
        var castedParameterValue = Convert.ToInt32(stringParameterValue);
    }
}
```

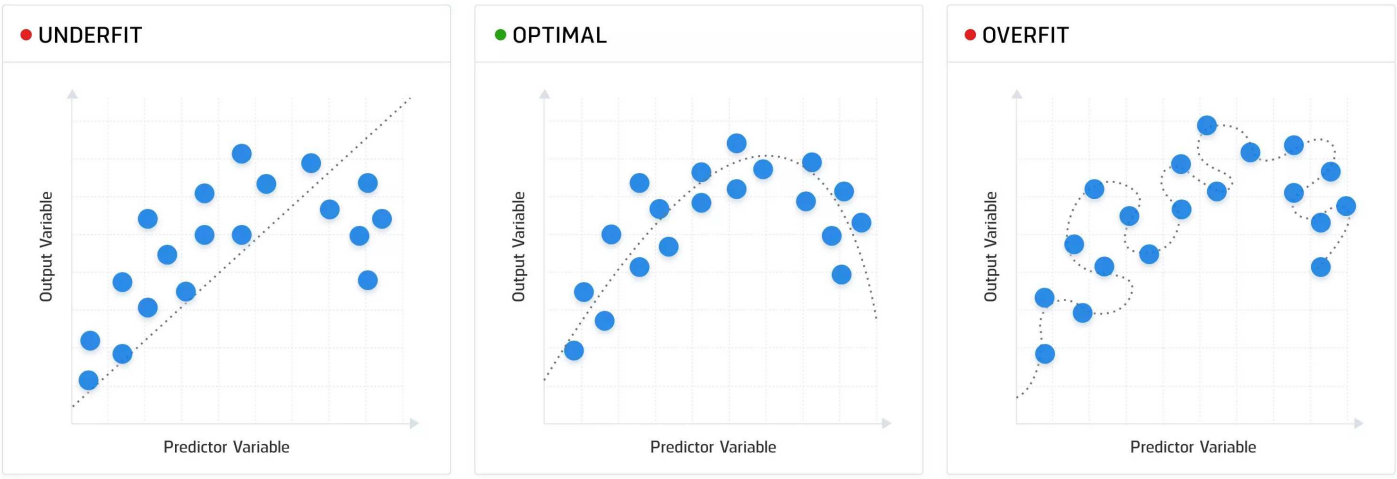
Alternatively, you can use the `Parameter(name)` attribute on top of class fields or properties to set their values. If there are no parameters in your project that match the name you pass to the attribute and you provide a default value to the method, it returns the default value.

```C#
// Use the Parameter attribute to set field values externally, enabling flexible configuration with default fallback.
public class ParameterizedAlgorithm : QCAlgorithm
{
    [Parameter("intParameterName")]
    public int IntParameterValue = 100;

    [Parameter("doubleParameterName")]
    public double DoubleParameterValue = 0.95;

    [Parameter("decimalParameterName")]
    public decimal DecimalParameterValue = 0.05m;
}
```

The parameter values are sent to your algorithm when you deploy the algorithm, so it's not possible to change the parameter values while the algorithm runs.

Overfitting

Overfitting occurs when a function is fit too closely fit to a limited set of training data. Overfitting can occur in your trading algorithms if you have many parameters or select parameters values that worked very well in the past but are sensitive to small changes in their values. In these cases, your algorithm will likely be fine-tuned to fit the detail and noise of the historical data to the extent that it negatively impacts the live performance of your algorithm. The following image shows examples of underfit, optimally-fit, and overfit functions:



An algorithm that is dynamic and generalizes to new data is more likely to survive across different market conditions and apply to other markets.

Look-Ahead Bias

Look-ahead bias occurs when an algorithm makes decisions using data that would not have yet been available. For instance, in optimization jobs, you optimize a set of parameters over a historical backtesting period. After the optimizer finds the optimal parameter values, the backtest period becomes part of the in-sample data. If you run a backtest over the same period using the optimal parameters, look-ahead bias has seeped into your research. In reality, it would not be possible to know the optimal parameters during the testing period until after the testing period is over. To avoid issues with look-ahead bias, optimize on older historical data and test the optimal parameter values on recent historical data. Alternatively, apply walk forward optimization to optimize the parameters on smaller batches of history.

Live Trading Considerations

To update parameters in live mode, add a Schedule Event that downloads a remote file and uses its contents to update the parameter values.

```C#
private Dictionary _parameters = new();
public override void Initialize()
{
    if (LiveMode)
    {
        Schedule.On(
            DateRules.EveryDay(),
            TimeRules.Every(TimeSpan.FromMinutes(1)),
            ()=>
            {
                var content = Download(urlToRemoteFile);
                // Convert content to _parameters
            });
    }
}
```

Examples

The following examples demonstrate common practices for using optimization parameters.

Example 1: Bond Candidate Selection

The following algorithm holds a 60/40 stock-bond portfolio rebalancing monthly. To select the best bond ETF to act as a proxy to trade bonds, we create a list of bond tickers and grid search the best using each to backtest. Using a `bond-id` parameter, which is an integer ranging from 0 to the list size - 1 with a step size of 1, we can perform optimization to test each of them.

```C#
public class OptimizationParameterAlgorithm : QCAlgorithm
{
    // A list of bond ETFs candidates to trade as bond proxies.
    private static readonly List<string> _bondList = new()
    {
        "TLT", "IEF", "SHY", "VCSH", "VCLT"
    };
    private Symbol _spy, _bond;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Get the bond candidate to be tested.
        var bondId = GetParameter("bond-id", 0);

        // Request SPY and the selected bond data for trading.
        _spy = AddEquity("SPY", Resolution.Minute).Symbol;
        _bond = AddEquity(_bondList[bondId], Resolution.Minute).Symbol;

        // Rebalance for a stock-bond portfolio monthly.
        Schedule.On(
            DateRules.MonthStart(_spy),
            TimeRules.AfterMarketOpen(_spy, 1),
            Rebalance
        );
    }

    private void Rebalance()
    {
        // Hold an all-weather stock-bond 60/40 portfolio.
        SetHoldings(_spy, 0.6m);
        SetHoldings(_bond, 0.4m);
    }
}
```

Example 2: Test Different Period

This algorithm holds SPY for the whole day following the direction from the previous day. To test which period is the most effective for this momentum-following strategy, we can set an `offset` parameter and adjust the backtest period accordingly.

You can also load the parameter values with the `Parameter` attribute.

```C#
[Parameter("offset")]
private int _offset = 0;
```

```C#
public class OptimizationParameterAlgorithm : QCAlgorithm
{
    // Get the number of months to offset the start date.
    [Parameter("offset")]
    private int _offset = 0;

    private Symbol _spy;
    private decimal _lastPrice = 0m;

    public override void Initialize()
    {
        // Create a variable start date with offset to test which periods the algorithm performs better.
        var startDate = new DateTime(2024, 1, 1);
        SetStartDate(startDate.AddMonths(_offset));
        // One-month fixed period test per backtest.
        SetEndDate(StartDate.AddMonths(1));

        // Request daily SPY data for trading.
        _spy = AddEquity("SPY", Resolution.Daily).Symbol;

        SetWarmUp(1);
    }

    public override void OnData(Slice slice)
    {
        // Trade based on updated data.
        if (slice.Bars.TryGetValue(_spy, out var bar))
        {
            // Trade with riding on the last day's momentum.
            if (_lastPrice != 0m && bar.Close > _lastPrice && !Portfolio[_spy].IsLong)
            {
                SetHoldings(_spy, 0.5m);
            }
            else if (_lastPrice != 0m && bar.Close < _lastPrice && !Portfolio[_spy].IsShort)
            {
                SetHoldings(_spy, -0.5m);
            }

            _lastPrice = bar.Close;
        }
    }
}
```

Example 3: Hyperparameter Tuning

The following example algorithm demonstrates an EMA cross strategy. To optimize for the best setup to attain the greatest Sharpe Ratio, the algorithm grid searches the best hyperparameters (the EMA periods) that do so.

C#

```
public class ParameterizedAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private ExponentialMovingAverage _fast;
    private ExponentialMovingAverage _slow;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        // For warming up indicators.
        Settings.AutomaticIndicatorWarmUp = true;

        // Request SPY data to feed the indicator and trade.
        _spy = AddEquity("SPY").Symbol;

        // Get and store the EMA settings to local parameters.
        var fastPeriod = GetParameter("ema-fast", 50);
        var slowPeriod = GetParameter("ema-slow", 200);

        // Set up the 2 EMA indicators for trade signal generation with the fast and slow periods from parameters.
        _fast = EMA("SPY", fastPeriod, Resolution.Daily);
        _slow = EMA("SPY", slowPeriod, Resolution.Daily);
    }

    public override void OnData(Slice slice)
    {
        if (slice.Bars.TryGetValue(_spy, out var bar))
        {
            // Following the uptrend indicated by price up in nearer periods.
            if (bar.Close > _fast && _fast > _slow)
            {
                SetHoldings(_spy, 0.5m);
            }
            // Following the downtrend indicated by price down in nearer periods.
            else if (bar.Close < _fast && _fast < _slow)
            {
                SetHoldings(_spy, -0.5m);
            }
            // Liquidate if the trend is not strong.
            else
            {
                Liquidate();
            }
        }
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
ParameterizedAlgorithm.cs C#

Optimization

Walk Forward Optimization

Introduction

Walk forward optimization is the practice of periodically adjusting the logic or parameters of a strategy to optimize some objective function over a trailing window of time. For example, say your strategy is to hold a long position when an asset is trading above its simple moving average (SMA) and to hold a short position when it's trading below the SMA. If your objective is to maximize returns, how many trailing data points should you use to compute the SMA?

Benefits

The parameter values that maximize your algorithm's performance are typically a function of the start and end dates of your training dataset. Market conditions change over time and a strategy that works well during one period may not perform as well during another. By continually adjusting the strategy parameters based on recent data, walk-forward optimization tailors your strategy to the latest market trends and patterns.

Optimization Frequency

There is a tradeoff when it comes to how often you optimize your algorithm's parameters. On one hand, if you frequently optimize your parameters, your parameters will be fit to the most recent, most meaningful market data. On the other hand, if you wait a long time between optimization sessions, your algorithm will run faster and you reduce the chance of overfitting. To define your optimization schedule, pass DateRules and TimeRules arguments to the train method.

EMA Crossover Example

Follow these steps to implement walk forward optimization for an exponential moving average (EMA) crossover strategy:

1. Define the backtest dates and starting cash.
2. Enable the automatic indicator warm up setting .
3. Subscribe to an asset .
4. Add some members for the EMA indicators.
5. Define the trading rules.
6. Define a method that will adjust the algorithm's behavior, given a set of optimal parameters.
7. Back in `initialize` method, generate all the possible combinations of parameter values.
8. Define the objective function.
9. Define the optimization function.
10. Schedule the optimization sessions.
11. Add a warm-up period .

    The warm-up period ensures that the algorithm calls the `_do_wfo` method at least one time before the algorithm starts trading.

For a full working example algorithm, see the following backtest:

Ideas

The following list provides some ideas on how you could use walk forward optimization:

1. Find the range of PE ratios that mazimize your algorithm's performance, and then use that range in your fundamental universe filter.
2. Experiment with asset universe diversity and counts.
3. Take 10 fundamental factors (PE, market cap, etc...). For each optimization session, step through and picks one factor at a time, using it as a ranking factor for the universe. Find the factor that leads to the greatest return over the lookback window and use it as the sole factor for the following month.

Machine Learning

Machine Learning

Key Concepts

Introduction

Machine learning is a field of study that combines statistics and computer science to build intelligent systems that predict outcomes. You can use machine learning techniques in your trading strategies.

Supported Libraries

LEAN supports several machine learning libraries. You can import these packages and use them in your algorithms.

Name	Version	Language	Import Statement	Exam
TensorFlow	2.19.1	Python	`import tensorflow`	
SciKit Learn	1.6.1	Python	`import sklearn`	
Py Torch	2.8.0	Python	`import torch`	
Keras	3.11.3	Python	`import keras`	
hmmlearn	0.3.3	Python	`import hmmlearn`	
tsfresh	0.20.2	Python	`import tsfresh`	
Stable-Baselines3	2.7.0	Python	`from stable_baselines3 import *`	
fastai	2.8.4	Python	`import fastai`	
Deap	1.4.3	Python	`import deap`	
XGBoost	3.0.5	Python	`import xgboost`	
mlfinlab	1.6.0	Python	`import mlfinlab`	
Accord	3.6.0	C#	`using Accord.MachineLearning;`	

Add New Libraries

To request a new library, contact us . We will add the library to the queue for review and deployment. Since the libraries run on our servers, we need to ensure they are secure and won't cause harm. The process of adding new libraries takes 2-4 weeks to complete. View the list of libraries currently under review on the Issues list of the Lean GitHub repository .

Save Models

After you train a model, you can save it into the Object Store . In QuantConnect Cloud, we back up your Object Store data on QuantConnect servers. In local algorithms, your local machine saves the Object Store data. If you save models in live algorithms, save them at the end of the training method so you can access the trained model again if your algorithm stops executing. If you save models in backtests, save them during the `OnEndOfAlgorithm` event handler so that saving multiple times doesn't slow down your backtest.

To view examples of storing library-specific models, see Popular Libraries .

Load Models

You can load machine learning models from the Object Store or a custom data file like pickle. If you load models from the Object Store, before you load the model into your algorithm, in the `Initialize` method, check if the Object Store already contains the model. To avoid look-ahead bias in backtests, don't train your model on the same data you use to test the model.

Library Errors

LEAN gracefully handles runtime errors that your code raises. It generally informs the location and cause of the error before the algorithm is terminated. FATAL ERROR EXCEPTION refers to a critical failure from third-party libraries, such as machine learning models, that LEAN doesn't handle.

Most QuantConnect members experience the invalid memory access error (SIGBUS) or division by zero error (SIGFPE) marked as "7 Killed" and "8 Killed" in the error message, respectively. The following sections explain some common errors and what causes them.

Memory Issues

You may experience the following memory issues:

- Out of Memory (OOM) : The process requires more memory (RAM or GPU memory) than is available, leading to a crash.
- Memory Leaks : If the program fails to properly release memoryy, it can eventually exhaust available resources.
- Invalid Memory Access : Bugs in custom C/C++ extensions, kernels, or libraries (for example, CUDA kernels for GPU acceleration) can cause misaligned memory access or invalid memory operations.

Numerical Instability

You may experience the following numerical instability errors:

- Division by Zero : This can occur due to input data with zeroes or during calculations, especially in custom loss functions or optimization algorithms.
- NaN or Infinity Values : These can arise due to data with NaN values, improper initialization, exploding gradients, or unstable mathematical operations.

Software Bugs

You may experience the following software bugs:

- Code Errors : Bugs in the implementation of the model, data preprocessing, or training loop can cause crashes.
- Library Incompatibility : Using incompatible versions of libraries (for example, TensorFlow or PyTorch) or dependencies can lead to fatal errors.

Machine Learning

Training Models

Introduction

Algorithms usually must process each timeslice within 10 minutes, but the `Train` method allows you to increase this time to train machine learning models. The length of time you can train depends on your training quotas.

Train Models

To train models immediately, call the `Train` method and pass in the name of your training method.

```
C#
# Run an ML model training session and extend method call timeouts.
Train(MyTrainingMethod);
```

Immediate training is most useful for training your model when you first deploy your strategy to production or when the model's performance begins to degrade.

Schedule Training Sessions

You can schedule model training sessions in a similar way to a Scheduled Event. To schedule a training session do this, pass in a DateRules and TimeRules argument to the `Train` method.

```
C#
// Set TrainingMethod to be executed at 8:00 am every Sunday
Train(DateRules.Every(DayOfWeek.Sunday), TimeRules.At(8, 0), MyTrainingMethod);
```

We recommend you schedule your training sessions for when the market is closed to get the best compute allocation. While the market is open, your CPU is occupied with processing incoming tick data and handling other LEAN events.

Training Quotas

Training resources are allocated with a leaky bucket algorithm where you can use a maximum of n-minutes in a single training session and the number of available minutes refills over time. This design gives you burst allocations when you need them and recharges the allowance to prepare for the next training.

Cloud Quotas

If you execute algorithms in QuantConnect Cloud, see Training Quotas for more information about the training quotas.

Local Quotas

If you execute algorithms locally, the following table shows the default settings for the leaky bucket algorithm:

Setting	Value
Capacity (minutes)	120
Time interval (minutes)	1440
Refill amount (minutes per time interval)	Capacity / 7

To allow virtually unlimited training for local algorithms, add the following key-value pairs to your Lean / Launcher / config.json file:

```
"scheduled-event-leaky-bucket-capacity" : 99999999,
"scheduled-event-leaky-bucket-time-interval-minutes" : 1,
"scheduled-event-leaky-bucket-refill-amount": 999999,
```

Check Model Readiness

In backtests, the `Train` method is synchronous, so it blocks your algorithm execution while the model trains. In live trading, the `Train` method is asynchronous, so ensure your model is trained before you continue the algorithm execution. Training occurs on a separate thread, so set a boolean flag to notify your algorithm of the model state. A semaphore is a thread-safe flag you can use to synchronize program operations across different threads.

C#

```
// Example of using a flag to signal the model is ready to use:
public class SemaphoreTrainingAlgorithm : QCAlgorithm
{
    // Initialize the model.
    private object _model;
    // Initialize the training state flag.
    private bool _modelIsTraining;

    public override void Initialize()
    {
        Train(MyTrainingMethod);
    }

    private void MyTrainingMethod()
    {
        _modelIsTraining = true;
        // Train the model here.
        _modelIsTraining = false;
    }

    public override void OnData(Slice slice)
    {
        // Don't use the model while it is training.
        if (_modelIsTraining)
        {
            return;
        }
        // Once training is complete, use the model.
        var result = _model.Predict();
    }
}
```

Examples

The following examples demonstrate some common practices for training the machine learning model.

Example 1: Scikit-Learn

The below algorithm makes use of `Scikit-Learn` library to predict the future price movement using the previous 5 OHLCV data.

The model is trained using rolling 2-year data. To ensure the model applicable to the current market environment, we recalibrate the model on every Sunday.

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
TrainingExampleAlgorithm.cs C#

Machine Learning

Popular Libraries

These are examples of using some of the most common machine learning libraries in an algorithm. Click one to learn more.

Aesera

Copula

GPlearn

Hmmlearn

Keras

PyTorch

Scikit-Learn

Tensorflow

Tslearn

XGBoost

Popular Libraries

Aesera

Introduction

This page explains how to build, train, deploy and store `Aesera` models.

To access Aesara, use the Legacy package environment. To set this environment, see the documentation in Cloud Platform or Local Platform .

Import Libraries

Import the `aesera` and `sklearn` libraries.

You need the `joblib` library to store models.

Create Subscriptions

In the `Initialize` method, subscribe to some data so you can train the `sklearn` model and make predictions.

Build Models

In this example, build a logistic regression prediction model that uses the following features and labels:

Data Category	**Description**
Features	Normalized daily close price of the SPY over the last 5 days
Labels	Return direction of the SPY over the next day

The following image shows the time difference between the features and labels:

Follow the below steps to build the model:

1. Initialize variables.
2. Construct the model graph.

3. Compile the model.

Train Models

You can train the model at the beginning of your algorithm and you can periodically re-train it as the algorithm executes.

Warm Up Training Data

You need historical data to initially train the model at the start of your algorithm. To get the initial training data, in the `Initialize` method, make a history request .

Define a Training Method

To train the model, define a method that fits the model with the training data.

Set Training Schedule

To train the model at the beginning of your algorithm, in the `Initialize` method, call the `Train` method.

To periodically re-train the model as your algorithm executes, in the `Initialize` method, call the `Train` method as a Scheduled Event .

Update Training Data

To update the training data as the algorithm executes, in the `OnData` method, add the current `TradeBar` to the `RollingWindow` that holds the training data.

Predict Labels

To predict the labels of new data, in the `OnData` method, get the most recent set of features and then call the `predict` method.

You can use the label prediction to place orders.

Save Models

Follow these steps to save `sklearn` models into the Object Store :

1. Set the key name you want to store the model under in the Object Store.
2. Call the `GetFilePath` method with the key.

   This method returns the file path where the model will be stored.
3. Call the `dump` method the file path.

   If you dump the model using the `joblib` module before you save the model, you don't need to retrain the model.

Load Models

You can load and trade with pre-trained `sklearn` models that you saved in the Object Store. To load a `sklearn` model from the Object Store, in the `Initialize` method, get the file path to the saved model and then call the `load` method.

The `ContainsKey` method returns a boolean that represents if the `model_key` is in the Object Store. If the Object Store does not contain the `model_key` , save the model using the `model_key` before you proceed.

Examples

The following examples demonstrate some common practices for using `Aesera` library.

Example 1: Aesera

The below algorithm makes use of `Aesera` library to predict the future price movement using the previous 5 OHLCV data. The model is trained using rolling 2-year data. To ensure the model applicable to the current market environment, we recalibrate the model on every Sunday.

Popular Libraries

Copula

Introduction

This page explains how to build, train, deploy and store `Copula` models.

Import Libraries

Import the `copula` , `scipy` , and `sklearn` libraries.

You need the `joblib` library to store models.

Create Subscriptions

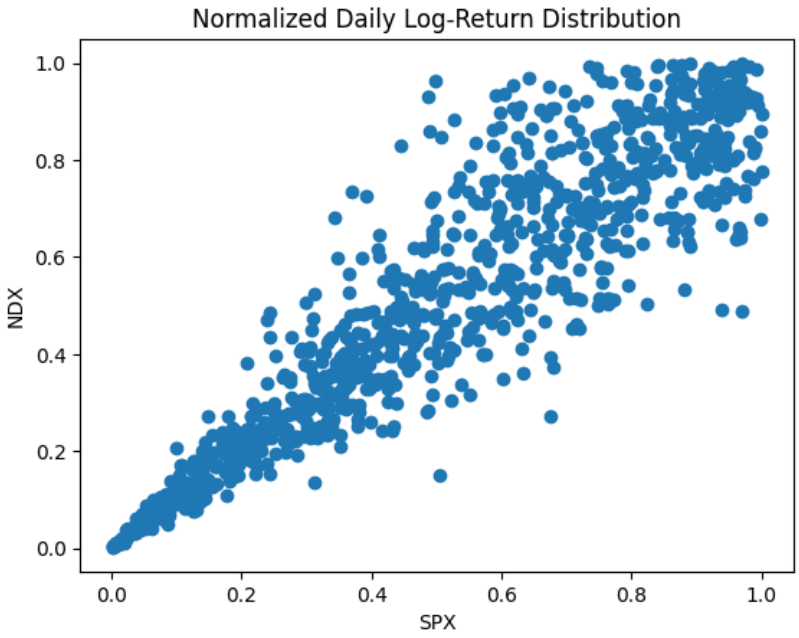
In the `Initialize` method, subscribe to some data so you can train the `copula` model and make predictions.

Since the algorithm is trading on options as speculation tools, to avoid insufficient buying power limited at the algorithm level, use `NullBuyingPowerModel` to bypass that.

Build Models

In this example, build a copula model to estimate the log-return bivariate distribution of the paired securities, such that it can predict the correlated future price movement more accurately.

The following image shows the 1-year daily log-return bivariate distribution (normalized) of SPX and NDX indices as an example. As shown in the plot, although they exhibit a certain level of correlation, the tail is fat in both side, making normal distribution assumption unsuitable. In such case, we can make use of distributions from the extreme value theorem (Frank, Gumbel, or Clayton).



Follow the below steps to build the model:

1. Create a helper class to handle the copula trading.
2. Create a method to transform the raw data into daily log-return, such that the series is more stationary while capturing

sufficient variance to accurately estimate the bivariate distribution.

3. Create a method to fit the copula with the distribution yielding the best log-likelihood.
4. (Optional) Reconfirm it by a 2-sample KS test.
5. Instantiate the helper class and fit the copula model.

Train Models

You can train the model at the beginning of your algorithm and you can periodically re-train it as the algorithm executes.

Warm Up Training Data

You need historical data to initially train the model at the start of your algorithm. To get the initial training data, in the `Initialize` method, make a history request .

Define a Training Method

To train the model, define a method that fits the model with the training data.

Set Training Schedule

To train the model at the beginning of your algorithm, in the `Initialize` method, call the `Train` method.

To periodically re-train the model as your algorithm executes, in the `Initialize` method, call the `Train` method as a Scheduled Event .

Update Training Data

To update the training data as the algorithm executes, make use of a consolidator to automatically update the rolling windows when a daily bar is consolidated through a consolidator handler.

Trading

In this example, we trade option mispricing arbitration using the copula model. More precisely, we are using the copula model to generate forward price series samples to price the options and do arbitration between options of the 2 correlated underlying. Since the copula can provide a distribution more accurately, we have an edge on the option pricing information.

Follow the below steps to trade the above logic:

1. Create a method to generate the forward price series samples of the 2 underlyings.
2. Create a method to price the options through averaging the payoff over all simulated paths.
3. Create a method to calculate the probability of winning (price converges to our model).

   We use the ratio of paths that the underlyings ever reach the strikes of the options, since ATM options are most efficiently priced and equal among all option pricing engine.

4. Create a method to calculate the aboves and output the mispriced option pairs.
5. Create a method to get the optimal position sizings for the top-expected-return trading opportunities.
6. Create a method to handle the portfolio and exit the positions when price converges.
7. Create a method in the base algorithm to open trades. Schedule its run.

   To simply implementation, this example will only open trades after the previous positions are all expired, but you can change it to scan by some time interval.

8. Scan every N minutes to exit the positions if the price discrepancy between a pair is converged.

Save Models

Follow these steps to save `copula` models into the Object Store :

1. Set the key name you want to store the model under in the Object Store.
2. Call the `GetFilePath` method with the key.

   This method returns the file path where the model will be stored.

3. Call the `dump` method the file path.

   If you dump the model using the `joblib` module before you save the model, you don't need to retrain the model.

Load Models

You can load and trade with pre-trained `copula` models that you saved in the Object Store. To load a `copula` model from the Object Store, in the `Initialize` method, get the file path to the saved model and then call the `load` method.

The `ContainsKey` method returns a boolean that represents if the `model_key` is in the Object Store. If the Object Store does not contain the `model_key` , save the model using the `model_key` before you proceed.

Examples

The below demonstrates the full algorithm of the above example.

Popular Libraries

GPlearn

Introduction

This page explains how to build, train, deploy and store `GPlearn` models.

To access GPlearn, use the Legacy package environment. To set this environment, see the documentation in Cloud Platform or Local Platform .

Import Libraries

Import the `gplearn` and `joblib` libraries.

You need the `joblib` library to store models.

Create Subscriptions

In the `Initialize` method, subscribe to some data so you can train the `GPLearn` model and make predictions.

Build Models

In this example, build a genetic programming feature transformation model and a genetic programming regression prediction model using the following features and labels:

Data Category	Description
Features	Daily percent change of the close price of the SPY over the last 5 days
Labels	Daily percent return of the SPY over the next day

The following image shows the time difference between the features and labels:

Follow these steps to create a method to build the model:

1. Declare a set of functions to use for feature engineering.

2. Call the `SymbolicTransformer` constructor with the preceding set of functions and then save it as a class variable.
3. Call the `SymbolicRegressor` constructor to instantiate the regression model.

Train Models

You can train the model at the beginning of your algorithm and you can periodically re-train it as the algorithm executes.

Warm Up Training Data

You need historical data to initially train the model at the start of your algorithm. To get the initial training data, in the `Initialize` method, make a history request .

Define a Training Method

To train the model, define a method that fits the model with the training data.

Set Training Schedule

To train the model at the beginning of your algorithm, in the `Initialize` method, call the `Train` method.

To periodically re-train the model as your algorithm executes, in the `Initialize` method, call the `Train` method as a Scheduled Event .

Update Training Data

To update the training data as the algorithm executes, in the `OnData` method, add the current close price to the `RollingWindow` that holds the training data.

Predict Labels

To predict the labels of new data, in the `OnData` method, get the most recent set of features and then call the `predict` method.

You can use the label prediction to place orders.

Save Models

Follow these steps to save `GPLearn` models into the Object Store :

1. Set the key names you want to store the models under in the Object Store.
2. Call the `GetFilePath` method with the keys.

   This method returns the file paths where the models will be stored.

3. Call the `dump` method the file paths.

   If you dump the models using the `joblib` module before you save the models, you don't need to retrain the models.

Load Models

You can load and trade with pre-trained `GPLearn` models that you saved in the Object Store. To load a `GPLearn` model from the Object Store, in the `Initialize` method, get the file path to the saved model and then call the `load` method.

The `ContainsKey` method returns a boolean that represents if `transformer_model_key` and `regressor_model_key` are in the Object Store. If the Object Store does not contain the keys, save the model using them before you proceed.

Examples

The following examples demonstrate some common practices for using `GPLearn` library.

Example 1: Genetic Learning

The below algorithm makes use of `GPLearn` library to predict the future price movement using the previous 5 OHLCV data. The model is trained using rolling 2-year data. To ensure the model applicable to the current market environment, we recalibrate the model on every Sunday.

Popular Libraries

Hmmlearn

Introduction

This page explains how to build, train, deploy and store `Hmmlearn` models.

Import Libraries

Import the `hmmlearn` and `joblib` libraries.

You need the `joblib` library to store models.

Create Subscriptions

In the `Initialize` method, subscribe to some data so you can train the `hmmlearn` model and make predictions.

Build Models

In this example, assume the market has only 2 regimes and the market returns follow a Gaussian distribution. Therefore, create a 2-component Hidden Markov Model with Gaussian emissions, which is equivalent to a Gaussian mixture model with 2 means.

To build the model, call the `GaussianHMM` constructor with the number of components, a covariance type, and the number of iterations:

Train Models

You can train the model at the beginning of your algorithm and you can periodically re-train it as the algorithm executes.

Warm Up Training Data

You need historical data to initially train the model at the start of your algorithm. To get the initial training data, in the `Initialize` method, make a history request .

Define a Training Method

To train the model, define a method that fits the model with the training data.

Set Training Schedule

To train the model at the beginning of your algorithm, in the `Initialize` method, call the `Train` method.

To periodically re-train the model as your algorithm executes, in the `Initialize` method, call the `Train` method as a Scheduled Event .

Update Training Data

To update the training data as the algorithm executes, in the `OnData` method, add the current close price to the `RollingWindow` that holds the training data.

Predict Labels

To predict the labels of new data, in the `OnData` method, get the most recent set of features and then call the `predict` method.

You can use the label prediction to place orders.

Save Models

Follow these steps to save `hmmlearn` models into the Object Store :

1. Set the key name you want to store the model under in the Object Store.
2. Call the `GetFilePath` method with the key.

   This method returns the file path where the model will be stored.

3. Call the `dump` method the file path.

   If you dump the model using the `joblib` module before you save the model, you don't need to retrain the model.

Load Models

You can load and trade with pre-trained `hmmlearn` models that you saved in the Object Store. To load a `hmmlearn` model from the Object Store, in the `Initialize` method, get the file path to the saved model and then call the `load` method.

The `ContainsKey` method returns a boolean that represents if the `model_key` is in the Object Store. If the Object Store does not contain the `model_key` , save the model using the `model_key` before you proceed.

Examples

The following examples demonstrate some common practices for using `Hmmlearn` library.

Example 1: Hidden Markov Model Regime Detection

The below algorithm makes use of `Hmmlearn` library to predict the future market regime through a hidden markov model using return data. If the regime is upward market, we hold SPY. Otherwise, we liquidate the portfolio. The model is trained using rolling 2-year data. To ensure the model applicable to the current market environment, we recalibrate the model on every Sunday.

Popular Libraries

Keras

Introduction

This page explains how to build, train, deploy and store `Keras` models.

Import Libraries

Import the `keras` libraries.

Create Subscriptions

In the `Initialize` method, subscribe to some data so you can train the `keras` model and make predictions.

Build Models

In this example, build a neural-network regression model that uses the following features and labels:

Data Category	Description
Features	Daily percent change of the open, high, low, close, and volume of the SPY over the last 5 days
Labels	Daily percent return of the SPY over the next day

The following image shows the time difference between the features and labels:

Follow the below steps to build the model:

1. In the `Initialize` method, create a `Sequential` object with several layers.

   Set the `input_shape` of the first layer to `(5, 5)` because each sample contains the percent change of 5 factors (percent change of the open, high, low, close, and volume) over the previous 5 days. Call the `Flatten` constructor because the input is 2-dimensional but the output is just a single value.

2. Call the `compile` method with a loss function, an optimizer, and a list of metrics to monitor.

Train Models

You can train the model at the beginning of your algorithm and you can periodically re-train it as the algorithm executes.

Warm Up Training Data

You need historical data to initially train the model at the start of your algorithm. To get the initial training data, in the `Initialize` method, make a history request .

Define a Training Method

To train the model, define a method that fits the model with the training data.

Set Training Schedule

To train the model at the beginning of your algorithm, in the `Initialize` method, call the `Train` method.

To periodically re-train the model as your algorithm executes, in the `Initialize` method, call the `Train` method as a Scheduled Event .

Update Training Data

To update the training data as the algorithm executes, in the `OnData` method, add the current `TradeBar` to the `RollingWindow` that holds the training data.

Predict Labels

To predict the labels of new data, in the `OnData` method, get the most recent set of features and then call the `predict` method.

You can use the label prediction to place orders.

Save Models

Follow these steps to save `keras` models into the Object Store :

1. Set the key name you want to store the model under in the Object Store.

   The key must end with a .keras extension for the native Keras format (recommended) or a .h5 extension.

2. Call the `GetFilePath` method with the key.

   This method returns the file path where the model will be stored.

3. Call the `save` method with the file path.

Load Models

You can load and trade with pre-trained `keras` models that you saved in the Object Store. To load a `keras` model from the Object Store, in the `Initialize` method, get the file path to the saved model and then call the `load_model` method.

The `ContainsKey` method returns a boolean that represents if the `model_key` is in the Object Store. If the Object Store does not contain the `model_key` , save the model using the `model_key` before you proceed.

Examples

The following examples demonstrate some common practices for using `Keras` library.

Example 1: Multi-layer Perceptron Model

The below algorithm makes use of `Keras` library to predict the future price movement using the previous 5 OHLCV data. The model is built with multi-layer perceptrons, ReLu activation function, and Adam optimization. It is trained using rolling 2-year data. To ensure the model applicable to the current market environment, we recalibrate the model on every Sunday.

Popular Libraries

PyTorch

Introduction

This page explains how to build, train, deploy and store `PyTorch` models.

Import Libraries

Import the `torch` and `joblib` libraries.

You need the `joblib` library to store models.

Create Subscriptions

In the `Initialize` method, subscribe to some data so you can train the `torch` model and make predictions.

Build Models

In this example, build a neural-network regression model that uses the following features and labels:

Data Category	**Description**
Features	The last 5 closing prices.
Labels	The following day's closing price

The following image shows the time difference between the features and labels:

Follow these steps to create a method to build the model:

1. Define a subclass of `nn.Module` to be the model.

   In this example, use the ReLU activation function for each layer.

2. Create an instance of the model and set its configuration to train on the GPU if it's available.

Train Models

You can train the model at the beginning of your algorithm and you can periodically re-train it as the algorithm executes.

Warm Up Training Data

You need historical data to initially train the model at the start of your algorithm. To get the initial training data, in the `Initialize` method, make a history request .

Define a Training Method

To train the model, define a method that fits the model with the training data.

Set Training Schedule

To train the model at the beginning of your algorithm, in the `Initialize` method, call the `Train` method.

To periodically re-train the model as your algorithm executes, in the `Initialize` method, call the `Train` method as a Scheduled Event .

Update Training Data

To update the training data as the algorithm executes, in the `OnData` method, add the current `TradeBar` to the `RollingWindow` that holds the training data.

Predict Labels

To predict the labels of new data, in the `OnData` method, get the most recent set of features and pass it to the model.

You can use the label prediction to place orders.

Save Models

Follow these steps to save `PyTorch` models into the Object Store :

1. Set the key name of the model to be stored in the Object Store.
2. Call the `GetFilePath` method with the key.

   This method returns the file path where the model will be stored.

3. Call the `dump` method the file path.

   If you dump the model using the `joblib` module before you save the model, you don't need to retrain the model.

Load Models

You can load and trade with pre-trained `PyTorch` models that you saved in the Object Store. To load a `PyTorch` model from the Object Store, in the `Initialize` method, get the file path to the saved model and then call the `load` method.

The `ContainsKey` method returns a boolean that represents if the `model_key` is in the Object Store. If the Object Store does not contain the `model_key` , save the model using the `model_key` before you proceed.

Examples

The following examples demonstrate some common practices for using `PyTorch` library.

Example 1: Multi-layer Perceptron Model

The below algorithm makes use of `PyTorch` library to predict the future price movement using the previous 5 OHLCV data. The

model is built with multi-layer perceptrons, ReLu activation function, and stochastic gradient descent optimization. It is trained using rolling 2-year data. To ensure the model applicable to the current market environment, we recalibrate the model on every Sunday.

Popular Libraries

Scikit-Learn

Introduction

This page explains how to build, train, deploy and store `Scikit-Learn` models.

Import Libraries

Import the `sklearn` and `joblib` libraries.

You need the `joblib` library to store models.

Create Subscriptions

In the `Initialize` method, subscribe to some data so you can train the `sklearn` model and make predictions.

Build Models

In this example, build a support vector regression prediction model that uses the following features and labels:

Data Category	Description
Features	Daily percent change of the open, high, low, close, and volume of the SPY over the last 5 days
Labels	Daily percent return of the SPY over the next day

The following image shows the time difference between the features and labels:

To build the model, call the `GridSearchCV` constructor with the SVR model, the parameter grid, a scoring method, the number of cross-validation folds:

Train Models

You can train the model at the beginning of your algorithm and you can periodically re-train it as the algorithm executes.

Warm Up Training Data

You need historical data to initially train the model at the start of your algorithm. To get the initial training data, in the `Initialize` method, make a history request .

Define a Training Method

To train the model, define a method that fits the model with the training data.

Set Training Schedule

To train the model at the beginning of your algorithm, in the `Initialize` method, call the `Train` method.

To periodically re-train the model as your algorithm executes, in the `Initialize` method, call the `Train` method as a Scheduled Event .

Update Training Data

To update the training data as the algorithm executes, in the `OnData` method, add the current `TradeBar` to the `RollingWindow` that holds the training data.

Predict Labels

To predict the labels of new data, in the `OnData` method, get the most recent set of features and then call the `predict` method.

You can use the label prediction to place orders.

Save Models

Follow these steps to save `sklearn` models into the Object Store :

1. Set the key name you want to store the model under in the Object Store.
2. Call the `GetFilePath` method with the key.

   This method returns the file path where the model will be stored.

3. Call the `dump` method the file path.

   If you dump the model using the `joblib` module before you save the model, you don't need to retrain the model.

Load Models

You can load and trade with pre-trained `sklearn` models that you saved in the Object Store. To load a `sklearn` model from the Object Store, in the `Initialize` method, get the file path to the saved model and then call the `load` method.

The `ContainsKey` method returns a boolean that represents if the `model_key` is in the Object Store. If the Object Store does not contain the `model_key` , save the model using the `model_key` before you proceed.

Examples

The following examples demonstrate some common practices for using Scikit-Learn library.

Example 1: Scikit-Learn

The below algorithm makes use of `Scikit-Learn` library to predict the future price movement using the previous 5 OHLCV data. The model is trained using rolling 2-year data. To ensure the model applicable to the current market environment, we recalibrate the model on every Sunday.

Popular Libraries

Tensorflow

Introduction

This page explains how to build, train, deploy and store `Tensorflow` models.

Import Libraries

Import the `tensorflow` library.

Create Subscriptions

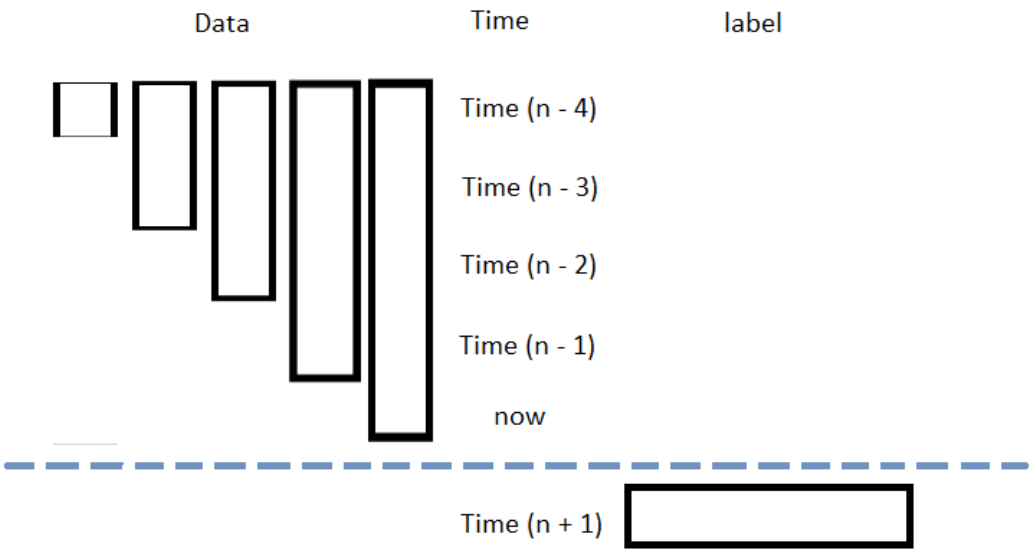
In the `Initialize` method, subscribe to some data so you can train the `tensorflow` model and make predictions.

Build Models

In this example, build a neural-network regression prediction model that uses the following features and labels:

Data Category	Description
Features	The last 5 close price differencing compared to current price
Labels	The following day's price change

The following image shows the time difference between the features and labels:



Follow these steps to create a method to build the model:

1. Define some parameters, including the number of factors, neurons, and epochs.
2. Create the model using built-in Keras API.

Train Models

You can train the model at the beginning of your algorithm and you can periodically re-train it as the algorithm executes.

Warm Up Training Data

You need historical data to initially train the model at the start of your algorithm. To get the initial training data, in the `Initialize` method, make a history request .

Define a Training Method

To train the model, define a method that fits the model with the training data.

Set Training Schedule

To train the model at the beginning of your algorithm, in the `Initialize` method, call the `Train` method.

To periodically re-train the model as your algorithm executes, in the `Initialize` method, schedule some training sessions .

Update Training Data

To update the training data as the algorithm executes, in the `OnData` method, add the current close price to the `RollingWindow` that holds the training data.

Predict Labels

To predict the labels of new data, in the `OnData` method, get the most recent set of features and then call the `run` method with new features.

You can use the label prediction to place orders.

Save Models

Follow these steps to save `Tensorflow` models into the Object Store :

1. Set the key name of the model to be stored in the Object Store.

   Note that the model has to have the suffix `.keras` .

2. Call the `GetFilePath` method with the key.

   This method returns the file path where the model will be stored.

3. Call the `save` method with the model and file path.
4. Save the model to the file path.

Load Models

You can load and trade with pre-trained `tensorflow` models that you saved in the Object Store. To load a `tensorflow` model from the Object Store, in the `Initialize` method, get the file path to the saved model and then recall the graph and weights of the model.

The `ContainsKey` method returns a boolean that represents if the `model.keras` is in the Object Store. If the Object Store doesn't contain the keys, save the model using them before you proceed.

Examples

The following examples demonstrate some common practices for using `Keras` library.

Example 1: Multi-layer Perceptron Model

The below algorithm makes use of `Keras` library to predict the future price movement using the previous 5 OHLCV data. The model is built with multi-layer perceptrons, ReLu activation function, and Adam optimization. It is trained using rolling 2-year

data. To ensure the model applicable to the current market environment, we recalibrate the model on every Sunday.

Popular Libraries

Tslearn

Introduction

This page explains how to build, train, deploy and store `Tslearn` models.

Import Libraries

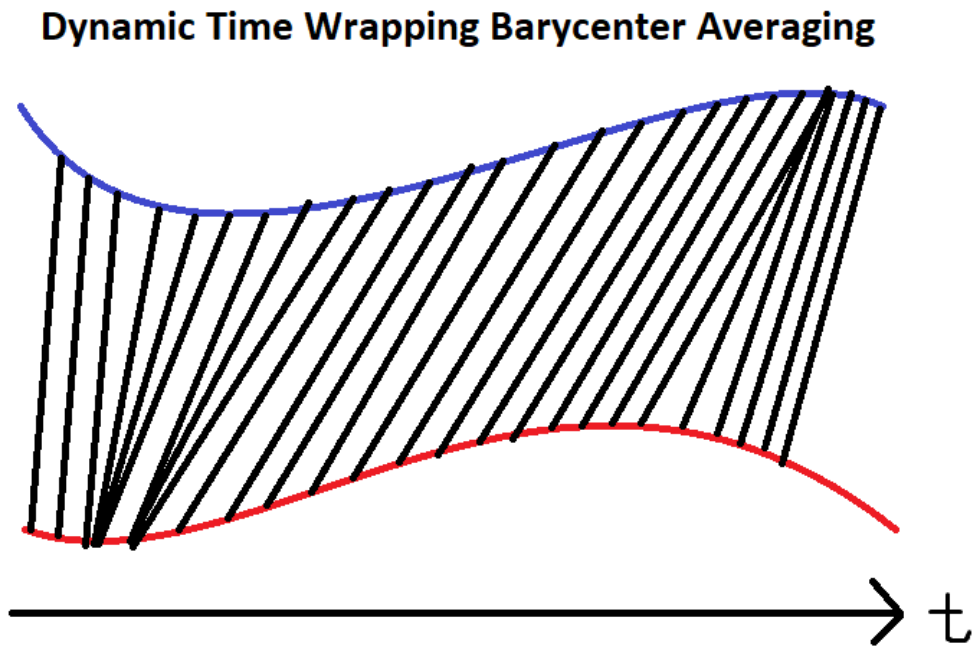
Import the `tslearn` libraries.

Create Subscriptions

In the `Initialize` method, subscribe to some data so you can train the `tslearn` model.

Build Models

In this example, train a model that clusters the universe of Equities into distinct groups and then allocate an equal portion of the portfolio to each cluster. To cluster the securities, instead of using a real-time comparison, apply Dynamic Time Wrapping Barycenter Averaging (DBA) to their historical prices and then run a k-means clustering algorithm. DBA is a technique of averaging a few time-series into a single one without losing much of their information. Since not all time-series move efficiently like in ideal EMH assumption, this technique allows similarity analysis of different time-series with sticky lags. The following image shows a visualization of the process. For more information about the technical details, see Dynamic Time Warping in the tslearn documentation .



To perform DBA and then cluster the securities by k-means, create a TimeSeriesKMeans model:

Train Models

You can train the model at the beginning of your algorithm and you can periodically re-train it as the algorithm executes.

Warm Up Training Data

You need historical data to initially train the model at the start of your algorithm. To get the initial training data, in the `Initialize` method, make a history request .

Define a Training Method

To train the model, define a method that fits the model with the training data.

Set Training Schedule

To train the model at the beginning of your algorithm, in the `Initialize` method, call the `Train` method.

To periodically re-train the model as your algorithm executes, in the `Initialize` method, call the `Train` method as a Scheduled Event .

Update Training Data

To update the training data as the algorithm executes, in the `OnData` method, add the current `TradeBar` to the `RollingWindow` that holds the training data.

Predict Labels

To predict the labels of new data, in the `OnData` method, get the most recent set of features and then call the `predict` method.

You can use the label prediction to place orders.

Save Models

Follow these steps to save `tslearn` models into the Object Store :

1. Set the key name of the model to be stored in the Object Store.
2. Call the `GetFilePath` method with the key.

   This method returns the file path where the model will be stored.

3. Delete the current file to avoid a `FileExistsError` error when you save the model.
4. Call the `to_hdf5` method with the file path.

Load Models

You can load and trade with pre-trained `tslearn` models that saved in Object Store. To load a `tslearn` model from the Object Store, in the `Initialize` method, get the file path to the saved model and then call the `from_hdf5` method.

The `ContainsKey` method returns a boolean that represents if the `model_key` is in the Object Store. If the Object Store does not contain the `model_key` , save the model using the `model_key` before you proceed.

Examples

The following examples demonstrate some common practices for using `Tslearn` library.

Example 1: Dynamic Time Wrapping (DTW) Clustering

The below algorithm makes use of `Tslearn` library to cluster stocks based on their recent year's price movement through DTW Barycenter Averaging. Then, we invest each cluster equally, while investing stocks within cluster equally as well. To ensure the model applicable to the current market environment, we recalibrate the model on every Sunday.

Popular Libraries

XGBoost

Introduction

This page explains how to build, train, deploy and store `XGBoost` models.

Import Libraries

Import the `XGBoost` and `joblib` libraries.

You need the `joblib` library to store models.

Create Subscriptions

In the `Initialize` method, subscribe to some data so you can train the `xgboost` model and make predictions.

Build Models

In this example, build a gradient boost tree regression prediction model that uses the following features and labels:

Data Category	Description
Features	The last 5 closing prices
Labels	The following day's closing price

The following image shows the time difference between the features and labels:

Train Models

You can train the model at the beginning of your algorithm and you can periodically re-train it as the algorithm executes.

Warm Up Training Data

You need historical data to initially train the model at the start of your algorithm. To get the initial training data, in the `Initialize`

method, make a history request .

Define a Training Method

To train the model, define a method that fits the model with the training data.

Set Training Schedule

To train the model at the beginning of your algorithm, in the `Initialize` method, call the `Train` method.

To periodically re-train the model as your algorithm executes, in the `Initialize` method, call the `Train` method as a Scheduled Event .

Update Training Data

To update the training data as the algorithm executes, in the `OnData` method, add the current `TradeBar` to the `RollingWindow` that holds the training data.

Predict Labels

To predict the labels of new data, in the `OnData` method, get the most recent set of features and then call the `predict` method.

You can use the label prediction to place orders.

Save Models

Follow these steps to save `xgboost` models into the Object Store :

1. Set the key name of the model to be stored in the Object Store.
2. Call the `GetFilePath` method with the key.

   This method returns the file path where the model will be stored.

3. Call the `dump` method the file path.

   If you dump the model using the `joblib` module before you save the model, you don't need to retrain the model.

Load Models

You can load and trade with pre-trained `xgboost` models that saved in Object Store. To load a `xgboost` model from the Object Store, in the `Initialize` method, get the file path to the saved model and then call the `load` method.

The `ContainsKey` method returns a boolean that represents if the `model_key` is in the Object Store. If the Object Store does not contain the `model_key` , save the model using the `model_key` before you proceed.

Examples

The following examples demonstrate some common practices for using `XGBoost` library.

Example 1: Gradient Boosting Model

The below algorithm makes use of `XGBoost` library to predict the future price movement using the previous 5 OHLCV data. The model is trained using rolling 2-year data. To ensure the model applicable to the current market environment, we recalibrate the model on every Sunday.

Machine Learning

Hugging Face

Hugging Face

Key Concepts

Introduction

Hugging Face is a leading company in natural language processing (NLP), providing powerful tools and pre-trained models that can significantly enhance your trading algorithms on QuantConnect. By integrating Hugging Face's models, you can easily leverage advanced NLP capabilities for tasks such as sentiment analysis, news classification, and market forecasting without defining and training the model yourself. Hugging Face provides an easy way to leverage the research of community members to increase the sophistication of your trading algorithms.

Supported Models

Hundreds of thousands of Hugging Face models are publicly available and ready to use. To view all of them, see the Model Hub on the Hugging Face website. The Model card tab of each model repository explains an overview of how the model works, its requirements, and a quick start guide.

QuantConnect Cloud caches some of the most popular models to speed up your development workflow. The following table shows the cached models:

Name	Category	Example
ahmedrachid/FinancialBERT-Sentiment-Analysis	Text Classification	
amazon/chronos-t5-base	Time Series Forecasting	
amazon/chronos-t5-large	Time Series Forecasting	
amazon/chronos-t5-small	Time Series Forecasting	
amazon/chronos-t5-tiny	Time Series Forecasting	
autogluon/chronos-bolt-base	Time Series Forecasting	
autogluon/chronos-t5-base	Time Series Forecasting	
autogluon/chronos-t5-large	Time Series Forecasting	
autogluon/chronos-t5-tiny	Time Series Forecasting	

AutonLab/MOMENT-1-large	Time Series Forecasting	
AventIQ-AI/sentiment-analysis-for-stock-market-sentiment	Time Series Forecasting	
bardsai/finance-sentiment-fr-base	Text Classification	
cardiffnlp/twitter-roberta-base-sentiment-latest	Text Classification	
deepseek-ai/DeepSeek-R1-Distill-Llama-70B	Text Generation	
distilbert/distilbert-base-cased-distilled-squad	Question Answering	
distilbert/distilbert-base-uncased	Fill-Mask	
FacebookAI/roberta-base	Fill-Mask	
google-bert/bert-base-uncased	Fill-Mask	
google/gemma-7b	Text Generation	
ibm-granite/granite-timeseries-ttm-r1	Time Series Forecasting	
microsoft/deberta-base	Fill-Mask	
mrm8488/distilroberta-finetuned-financial-news-sentiment-analysis	Text Classification	
nickmuchi/deberta-v3-base-finetuned-finance-text-classification	Text Classification	
nickmuchi/distilroberta-finetuned-financial-text-classification	Text Classification	
nickmuchi/sec-bert-finetuned-finance-classification	Text Classification	
openai-community/gpt2	Text Generation	
ProsusAI/finbert	Text Classification	
Salesforce/moirai-1.0-R-base	Time Series Forecasting	
Salesforce/moirai-1.0-R-large	Time Series Forecasting	
Salesforce/moirai-1.0-R-small	Time Series Forecasting	
StephanAkkerman/FinTwitBERT-sentiment	Text Classification	
yiyanghkust/finbert-tone	Text Classification	

To see the commit hash of the cached models, run the following algorithm in QC Cloud and then view the logs :

Train Models

Hugging Face models are pre-trained, so you can quickly load them into your algorithms and use their output to inform your trading decisions. To customize the model for your specific use case, you can fine-tune the pre-trained model with your own training data.

Examples

You may refer to the following table for examples of popular Hugging Face models:

Model Name	**Model Type**	**Example**
Chronos-T5	Pretrained Time Series Forecasting Transformer Model	Example
FinBERT	Text Classification Transformer Model	Example

Hugging Face

Popular Models

These are examples of using some of the most common Hugging Face models in an algorithm. Click one to learn more.

Chronos-T5

Time series forecasting

DistilBERT

Question Answering

FinBERT

Sentiment analysis

Popular Models

Chronos-T5

Introduction

This page explains how to use Chronos-T5 in LEAN trading algorithms. The model repository provides the following description:

> Chronos is a family of **pretrained time series forecasting models** based on language model architectures. A time series is transformed into a sequence of tokens via scaling and quantization, and a language model is trained on these tokens using the cross-entropy loss. Once trained, probabilistic forecasts are obtained by sampling multiple future trajectories given the historical context. Chronos models have been trained on a large corpus of publicly available time series data, as well as synthetic data generated using Gaussian processes.
>
> For details on Chronos models, training data and procedures, and experimental results, please refer to the paper Chronos: Learning the Language of Time Series .

Use Cases

The Chronos-T5 model is a time series forecasting model. The following use cases explain how you might utilize it in trading algorithms:

- Forecast the future equity curves for a set of assets, then pass them to an optimizer to determine the weights that maximize the future Sharpe ratio of the portfolio.
- Forecast the future volatility of an asset to manage risk and opimize portfolio allocations.
- Train the model to forecast the impact of specific events on an asset and then adjust your holdings in response to the event. For example, you could train the model to forecast the impact of news , corporate actions , or financial reports .

Load Pre-Trained Model

Follow these steps to load the pre-trained Chronos-T5 model:

1. Add the following imports to the top of your code file:
2. Call the `ChronosPipeline.from_pretrained` method with the model path.

   In QuantConnect Cloud, the path to the tiny model is amazon / chronos-t5-tiny .

3. (Optional) Set the seed to enable reproducibility.

Fine-Tune Model

The Chronos-T5 model is pre-trained, so you don't need to fine-tune it. Fine-tuning the model just tailors it to your specific use case. Follow these steps to fine-tune it:

1. Add the following imports to the top of your code file:
2. In the `initialize` method, define the model and some of its settings.

   In QuantConnect Cloud, the path to the tiny model is amazon/chronos-t5-tiny .

3. Define the fine-tuning method.
4. Create DataFrame(s) that contain your training samples.

Each DataFrame should have one column, named "target". The rows throughout the DataFrame should span a consistent time. If there is no data for the time step (for example, the weekend price of an Equity asset), put NaN as the value for the time step. The following example demonstrates how to create DataFrames that contain one year of trailing prices for a set of Equities:

For more information about history requests, see History Requests .

5. Call the fine-tuning method with the training data.
6. Load the fine-tuned model.

Forecast Time Series

Follow these steps to forecast time series with Chronos-T5:

1. Get some historical data .
2. Load the pre-trained or fine-tuned model.
3. Forecast the future time series.

   In this example, the model returns several future price paths for each asset.

4. Aggregate the future prices paths of each asset into a single price path for each asset.

   For example, take the median value of each time step in the future price paths.

Examples

The following examples demonstrate usage of the Chronos-T5 model.

Example 1: Price Prediction

The following algorithm selects the most liquid assets at the beginning of each month. Once a quarter, it gets the trailing year of prices for all the assets in the universe and then forecasts the prices paths of all the assets over the upcoming quarter. It then uses the SciPy package to find the weights that maximize the future Sharpe ratio of the portfolio and rebalances the portfolio to those weights.

Example 2: Fine Tuning

The following algorithm expands the preceding algorithm by fine-tuning the model before each forecast:

Popular Models

DistilBERT

Introduction

This page explains how to use DistilBERT in LEAN trading algorithms. The model repository provides the following description:

> The DistilBERT model was proposed in the blog post Smaller, faster, cheaper, lighter: Introducing DistilBERT, adistilled version of BERT , and the paper DistilBERT, adistilled version of BERT: smaller, faster, cheaper and lighter . DistilBERT is a small, fast, cheap and light Transformer model trained by distilling BERT base. It has 40% less parameters than bert-base-uncased, runs 60% faster while preserving over 95% of BERT's performances as measured on the GLUE language understanding benchmark. This model is a fine-tune checkpoint of DistilBERT-base-cased , fine-tuned using (a second step of) knowledge distillation on SQuAD v1.1 .

Use Cases

The DistilBERT model is a question answering model. The following use cases explain how you might utilize it in trading algorithms:

- Summarize the upcoming changes reported through regulatory articles in preparation of sentiment analysis.
- Extract relevant information from news feeds like Tiingo and Benzinga while reducing noise.
- Parse through US SEC Filings to find critical information about company performance, legal issues, or changes in management.

Load Pre-Trained Model

Follow these steps to load the pre-trained DistilBERT model:

1. Import the pipeline.
2. Create a pipeline object with the DistilBERT model.

   In QuantConnect Cloud, the path is distilbert / distilbert-base-cased-distilled-squad .

Answer Questions

Follow these steps to answer questions with DistilBERT:

1. Load the model .
2. Define the context.
3. Create the question.
4. Pass the question and context to the pipeline.

   The following dictionary is an example result:

Examples

Popular Models

FinBERT

Introduction

This page explains how to use FinBERT in LEAN trading algorithms. The model repository provides the following description:

> FinBERT is a pre-trained NLP model to analyze sentiment of financial text. It is built by further training the BERT language model in the finance domain, using a large financial corpus and thereby fine-tuning it for financial sentiment classification. Financial PhraseBank by Malo et al. (2014) is used for fine-tuning. For more details, please see the paper FinBERT: Financial Sentiment Analysis with Pre-trained Language Models and our related blog post on Medium.
>
> The model will give softmax outputs for three labels: positive, negative or neutral.

Use Cases

The FinBERT model is a sentiment analysis model. The following use cases explain how you might utilize it in trading algorithms:

- Analyze the sentiment of the latest news articles for specific companies , then form a long-short portfolio with assets that have the most postive and most negative news sentiment.
- Monitor the sentiment of regulatory alerts in a risk management model and liquidate holdings when sentiment is extremely negative.
- Generate sentences based on information from other datasets and then feed them into the model to determine the sentiment. For example, you could use the US Government Contracts dataset to create the string "The Department of State grants AAPL a contract for the purchase of mobile phones".

Load Pre-Trained Model

Follow these steps to load the pre-trained FinBERT model:

1. Import the model and tokenizer classes.
2. Define the path were the model is stored.

   In QuantConnect Cloud, the path is ProsusAI / finbert .

3. Create a TFBertForSequenceClassification model.
4. Create a BertTokenizer object.
5. (Optional) Set the seed to enable reproducibility.

Fine-Tune Model

The FinBERT model is pre-trained, so you don't need to fine-tune it. Fine-tuning the model just tailors it to your specific use case. Follow these steps to fine-tune it:

1. Import the `Dataset` class.
2. Load the pre-trained model .
3. Compile the model with an optimizer and loss function.

   You only need to compile the model once. TensorFlow offers many other optimizer and loss classes you can use.

4. Create a DataFrame that contains your training samples.

The DataFrame should have two columns, named "text" and "label". The label must be an integer that represents the sentiment class. By default, the FinBERT model has three classes. Class zero represents negative sentiment, class one represents neutral sentiment, and class two represents positive sentiment.

5. Convert the samples DataFrame to a `tf.data.Dataset` object.
6. Tokenize the text in each training sample.
7. Call the model's prepare_tf_dataset method.
8. Call the model's fit method.

Analyze Sentiment

Follow these steps to analyze the sentiment of some text with FinBERT:

1. Load the model .
2. (Optional) Fine-tune the model .
3. Get the text you want the model to analyze.

   The model can analyze a single sentence or a list of sentences.

4. Tokenize the text(s).

   For more information about how to tokenize, see the PreTrainedTokenizer.__call__ reference on the Hugging Face website.

5. Perform dictionary unpacking on the preceding result and pass it to the model as input.
6. Apply softmax to the outputs to get probabilities of each sentiment class.

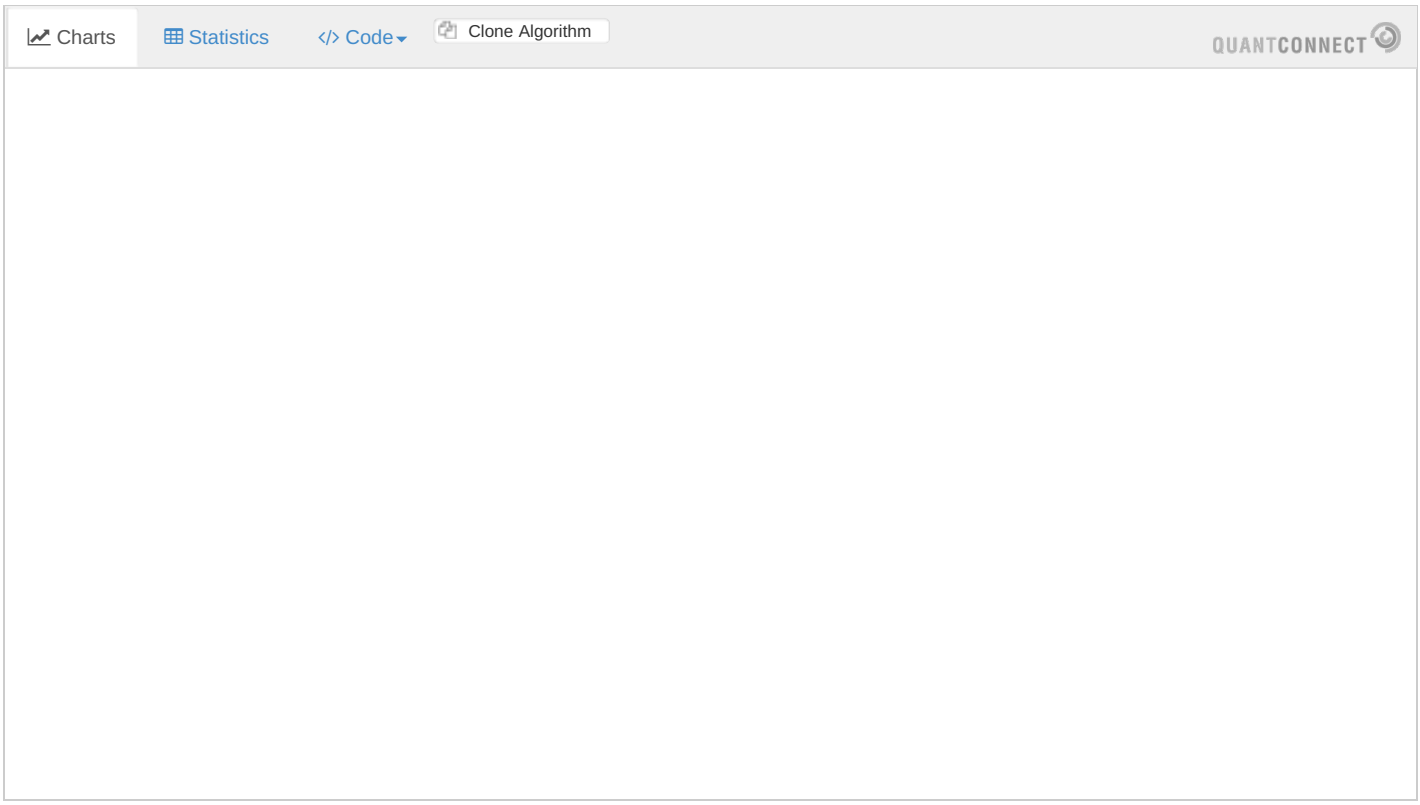
   The result of the preceding operation is a two dimensional numpy array. Each element in the two dimensional array is a list that contains the probability that the sentiment of the corresponding sentence is negative, neutral, or postiive, respectively. For example, you may get the following result if you use a single sentence as input. The result shows that the input is more positive than negative, but is likely neutral.

Examples

The following examples demonstrate usage of the FinBERT model.

Example 1: 10-day Sentiment Score

The following algorithm selects a volatile asset at the beginning of each month. It gets the Tiingo News articles that were released for the asset over the previous 10 days and then feeds them into the pre-trained FinBERT model. It then aggregates the sentiment scores of all the news releases. If the aggregated sentiment score is positive, it enters a long position for the month. If it's negative, it enters a short position for the month.



Example 2: 30-day Sentiment Score With Fine Tuning

The following algorithm selects a volatile asset at the beginning of each month. It gets the Tiingo News articles that were released for the asset over the previous 30 days to generate the training set. The label is the market return that occurs from the current news release to the next news release. The algorithm then fine-tunes the model, calculates the sentiment, and rebalances the portfolio.

These algorithms require a GPU node.

Algorithm Framework

Algorithm Framework

Overview

Introduction

Since QuantConnect was formed in 2013, the community has created more than 2 million algorithms. Although they all seek to achieve different results, they can all be abstracted into the same core features. The Algorithm Framework is our attempt to provide this well-defined structure to the community, encouraging good design and making your work more reusable. LEAN provides many pre-built Algorithm Framework models you can use in your algorithms.

Strategy Styles

The algorithm framework works best for portfolio rebalancing, ranking, and multi-factor style strategies. The default portfolio construction models are designed to weigh and invest in the assets as the signals values shift, aiming to be fully invested as much as possible.

Some strategies might find it difficult to fit this model. Technical entry signals are often tightly coupled with their exit criteria, they may not have defined hold periods, and they might follow the market to determine when to exit. This style of strategy can be addressed in a few ways:

1. Short Term Signals - Emitting insights for short periods until they are no longer valid.
2. Cancel Signals - Updating an insight's expiry time so that it's expired upon the exit signal.
3. Alpha Exit Signal - Treating exit as its own signal, a separate Alpha model could indicate exit.
4. Risk Model - Tracking exit using a risk model that attempts to lock in gains.

These techniques are still not ideal for many strategies. For example, with only a single signal, the built in `EqualWeightingPortfolioConstructionModel` will allocate 100% of buying power to a single asset, over concentrating risk. If a second signal comes a few days later, it would sell 50% of the portfolio to free cash and invest in the second asset. This can create high trading costs through portfolio churn.

With a custom portfolio construction model , signals can mean what you need for your specific strategy style. For example, you might create a " `MultiBetPortfolioConstructionModel` " that uses each insight to open a specific "bet" that is separately allocated capital and tracked.

Important Terminology

The following table defines important Algorithm Framework terminology:

Term	Description
Universe Selection model	A framework module that selects assets for your algorithm.
Alpha model	A framework module that generates trading signals on the assets in your universe.
Insight	An object that represents a trading signal. The Alpha model generates these objects for the Portfolio Construction model.
Portfolio Construction model	A framework module that determines position size targets based on the `Insight` objects it receives from the Alpha model.
PortfolioTarget	An object that represents the target position size for an asset in the universe. The Portfolio Construction model generates these objects for the Risk Management model.
Risk Management	A framework module that manages market risks and ensures the algorithm remains within target parameters. This model adjusts the `PortfolioTarget` objects it receives from the Portfolio Construction model before they reach the Execution model.
Execution	A framework module that places trades to reach the target risk-adjust portfolio based on the `PortfolioTarget` objects it receives from the Risk Management model.

Framework Advantages

We recommend you design strategies with the Algorithm Framework in most cases. The Algorithm Framework has many advantages over the classic algorithm design.

Pluggable Algorithm Modules

With the Algorithm Framework, your code can instantly utilize all modules built within the framework. The modules clip together in well-defined ways, which enable them to be shared and swapped interchangeably.

Focus On Your Strengths

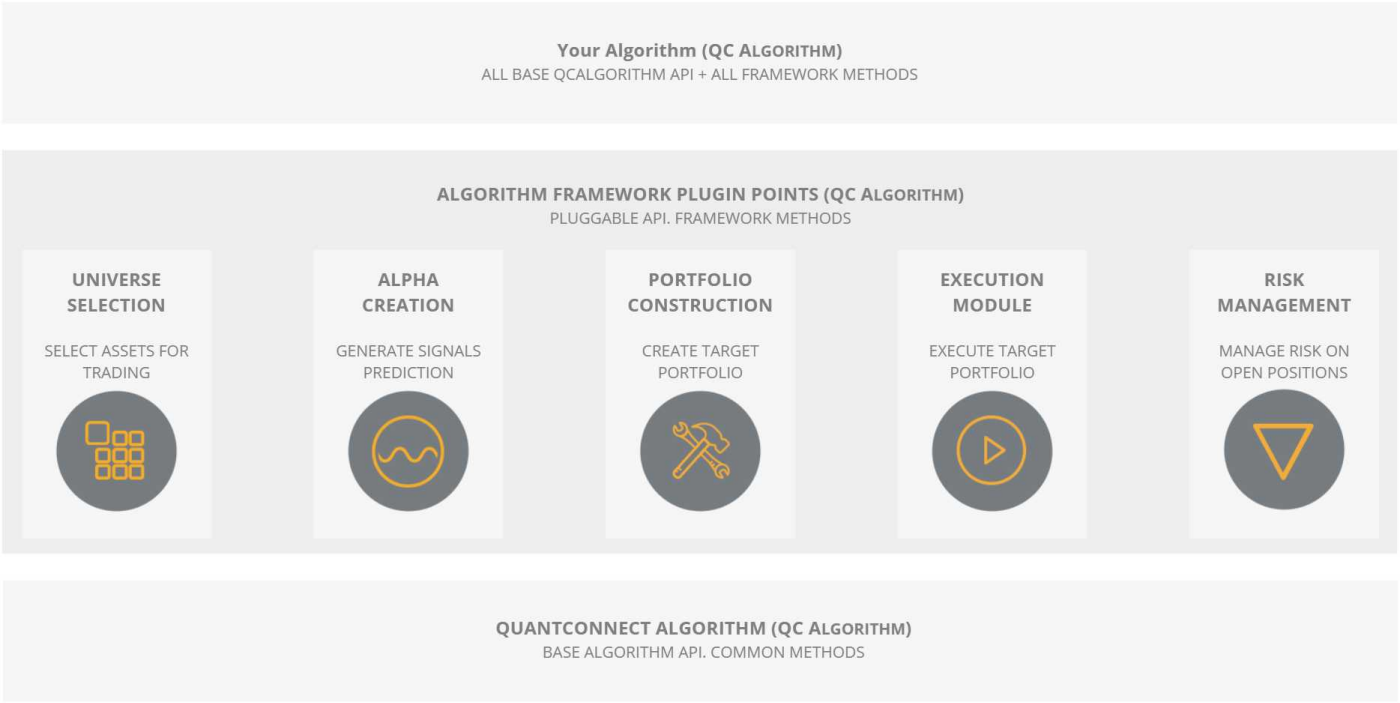
If you write code in modules, you can focus on your strengths. If you are great at building universes, you can build Universe Selection modules. If you have risk management experience, you could write reusable risk monitoring technology.

Reduce Development By Using Community Modules

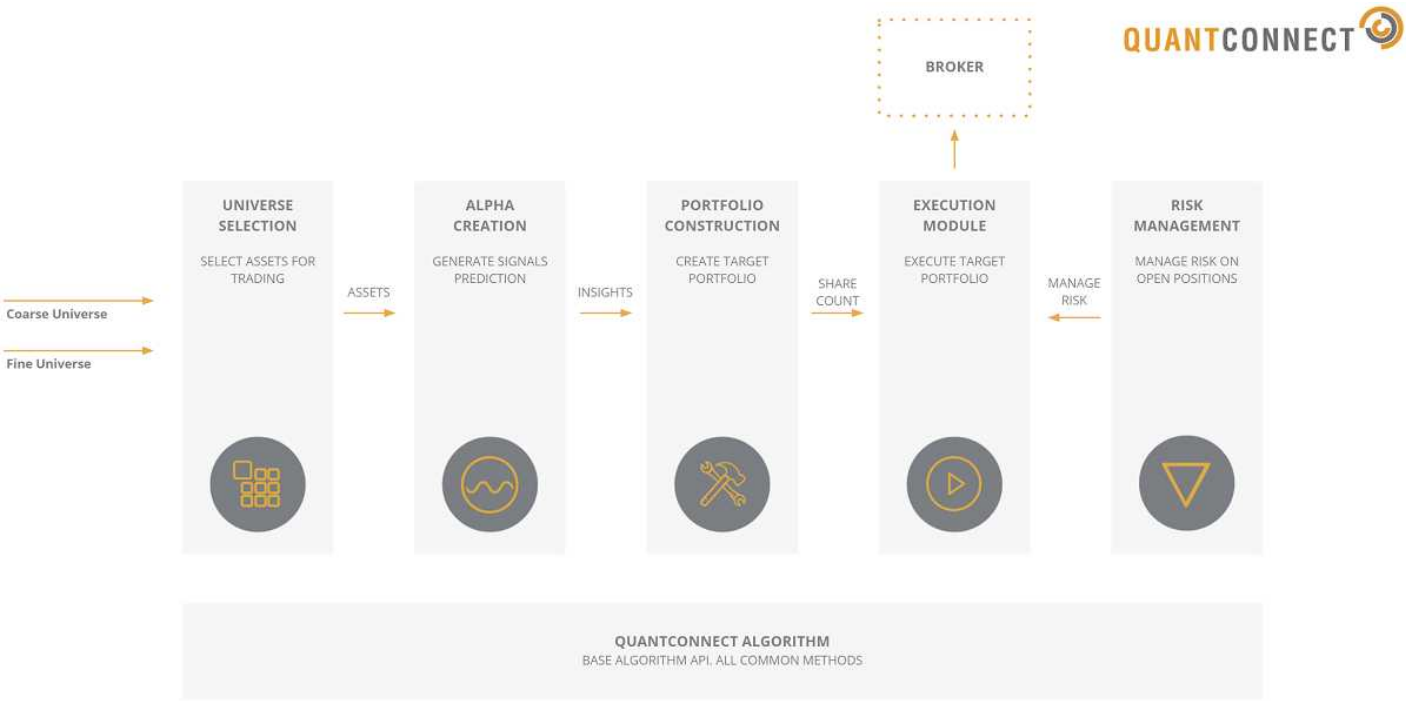
Easily share modules you've made between algorithms or pull in ones produced by the community. The strict separation of duties makes the Algorithm Framework perfect for reusing code between strategies.

System Architecture

The Algorithm Framework is built into the `QCAlgorithm` class, so your strategy can access all the normal methods you use for algorithm development. You should be able to copy and paste your algorithms across without any changes. You don't need a separate class.



The framework data output of each module flows into the following module. The assets that the Universe Selection model selects are fed into the Alpha model to generate trade signals (`Insight` objects). The `Insight` objects from the Alpha model are fed into the Portfolio Construction model to create `PortfolioTarget` objects, which contain the target number of units to hold for each asset. The `PortfolioTarget` objects from the Portfolio Construction model are fed into the Risk Management model to ensure the targets are within safe risk parameters and to adjust the `PortfolioTarget` objects if necessary. The `PortfolioTarget` objects from the Risk Management model are fed into the Execution model, which efficiently places trades to acquire the target portfolio.



```C#
public class MyFrameworkAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetUniverseSelection(new EmaCrossUniverseSelectionModel());
        AddAlpha(new RsiAlphaModel());
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
        SetExecution(new ImmediateExecutionModel());
        AddRiskManagement(new NullRiskManagementModel());
    }
}
```

For simple strategies, it may seem like overkill to abstract out your algorithm concepts. However, even simple strategies can benefit from reusing the ecosystem of modules available in QuantConnect. Imagine pairing your EMA-cross Alpha model with a better execution system or plugging in an open-source trailing stop Risk Management model .

Separation of Concerns

The separation of concerns principle is an algorithm design principle where individual components are isolated and their responsibilities don't overlap. The Algorithm Framework is designed to uphold this design principle. The individual framework components shouldn't rely on the state of the other framework components in order to operate.

In the Algorithm Framework, the models should not communicate. The universe constituents and the `Insight` objects you emit should be the same, regardless of what you select for the Portfolio Construction, Risk Management, and Execution models. If your algorithm logic naturally violates the separation of concerns design principle, it's more appropriate to use the classic or hybrid design .

Framework vs Classic Design

Select the classic or Algorithm Framework design based on the needs of your algorithm. If your algorithm uses special order types or passes information between the different algorithm components, use the classic design. If you can reuse existing modules and the Algorithm Framework modules can work in isolation, use the Algorithm Framework design.

Algorithm Framework

Universe Selection

Universe Selection

Key Concepts

Introduction

The Universe Selection model creates `Universe` objects, which select the assets for your algorithm. As the universe changes, we notify your algorithm through the `OnSecuritiesChanged` event handler. With this event handler, you can track the current universe constituents in other parts of your algorithm without breaking the separation of concerns design principle.

Types of Universe Selection

We have identified several types of universes that cover most people's requirements and built helper classes to make their implementation easier. The following table describes the types of pre-built Universe Selection models:

Universe Type	Description
Manual Universes	Universes that use a fixed, static set of assets
Fundamental Universes	Universes for US Equities that are based on coarse price or fundamental data
Scheduled Universes	Universes that trigger on regular, custom intervals
Futures Universes	Universes that subscribe to Future chains
Option Universes	Universes that subscribe to Option chains

Add Models

To add a Universe Selection model, in the `Initialize` method, call the `AddUniverseSelection` method.

```
C#
var symbols = new [] {QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA)};
AddUniverseSelection(new ManualUniverseSelectionModel(symbols));
```

To view all the pre-built Universe Selection models, see Types of Universe Selection .

Multi-Universe Algorithms

You can add multiple Universe Selection models to a single algorithm.

```
// Configures asynchronous universe settings for faster updates and includes both manual and EMA cross universe
selection models for dynamic and static selection.
UniverseSettings.Asynchronous = true;
var symbols = new [] {QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA)};
AddUniverseSelection(new ManualUniverseSelectionModel(symbols));

AddUniverseSelection(new EmaCrossUniverseSelectionModel());
```

If you add multiple Universe Selection models, the algorithm subscribes to the constituents of all the models.

Model Structure

Universe Selection models should extend the `UniverseSelectionModel` class. Extensions of the `UniverseSelectionModel` class must implement the `CreateUniverses` method, which receives an `algorithm` object and returns an array of `Universe` objects.

```
// Extend UniverseSelectionModel to define a custom security universe tailored to the algorithm.
public class MyUniverseSelectionModel : UniverseSelectionModel
{
    // Creates the universes for this algorithm
    public override IEnumerable<Universe> CreateUniverses(QCAlgorithm algorithm)
    {
        return new List<Universe>();
    }

    // Gets the next time the framework should invoke the `CreateUniverses` method to refresh the set of universes.
    public override DateTime GetNextRefreshTimeUtc()
    {
        return DateTime.MaxValue;
    }
}
```

The `algorithm` argument that the methods receive is an instance of the base `QCAlgorithm` class, not your subclass of it.

Generally, you should be able to extend one of the pre-built Universe Selection types. If you need to do something that doesn't fit into the pre-built types, let us know and we'll create a new foundational type of Universe Selection model.

Track Security Changes

When the Universe Selection model adjusts the universe constituents, we notify your algorithm through the `OnSecuritiesChanged` event handler on your other framework components. The method receives `QCAlgorithm` and `SecurityChanges` objects. The `QCAlgorithm` object is an instance of the base QCAlgorithm class, not a reference to your algorithm object. To access the added securities, check the `changes.AddedSecurities` method property. To access the removed securities, check the `changes.RemovedSecurities` method property.

```
// Log added and removed securities to update the universe and adapt the strategy.
public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
{
    foreach (var security in changes.AddedSecurities)
    {
        Log($"Added {security.Symbol}");
    }

    foreach (var security in changes.RemovedSecurities)
    {
        Log($"Removed {security.Symbol}");
    }
}
```

Live Trading Considerations

In live trading, the securities in your brokerage account are added to your user-defined universe. If your Universe Selection model doesn't select these securities, they are not removed from your user-defined universe. When you liquidate the positions, you can remove them from the user-defined universe. If you call the `RemoveSecurity` method, it automatically liquidates the position.

To see the securities in your user-defined universe that were loaded from your brokerage account, check your algorithm's ActiveSecurities in the OnWarmupFinished event handler.

Examples

The following examples demonstrate some common practices for implementing the universe selection model.

Example 1: Liquid Universe Selection Model

The following algorithm selects a universe of the top 20 liquid equity through the framework universe selection model. Then, buying and holding these popular equities equally to bet on their high demand will drive up the price.

```C#
public class FrameworkUniverseSelectionAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Rebalance weekly to allow time to capitalize on the trend
        UniverseSettings.Schedule.On(DateRules.WeekStart());
        // Since the selection is based on daily data, we only require daily data to trade daily.
        UniverseSettings.Resolution = Resolution.Daily;
        // Add universe selection model to select the most liquid stocks to trade with.
        AddUniverseSelection(new FundamentalUniverseSelectionModel(Selection));

        // Sent insights on buying and holding the most liquid stocks for a week.
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(7)));
        // Evenly dissipate the capital risk among selected stocks.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }

    private IEnumerable<Symbol> Selection(IEnumerable<Fundamental> fundamental)
    {
        // Select the top 20 liquid equity to trade the popularity, suggesting a higher demand to drive the price.
        return fundamental.OrderByDescending(x => x.DollarVolume)
            .Take(20)
            .Select(x => x.Symbol);
    }
}
```

Example 2: Multiple Universe Selection Models

The following algorithm implements two framework universe selection models to manually select the most liquid market ETFs and those with EMA crossing events to trade on trend changes. Then, buy and hold these equities equally to bet on the insights and dissipate the capital risk.

```
C#
public class FrameworkUniverseSelectionAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Rebalance weekly to allow time to capitalize on the trend
        UniverseSettings.Schedule.On(DateRules.WeekStart());
        // Since the selection is based on daily data, we only require daily data to trade daily.
        UniverseSettings.Resolution = Resolution.Daily;
        // Add a universe selection model to select the most liquid ETFs manually.
        var symbols = new [] {
            QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA),
            QuantConnect.Symbol.Create("QQQ", SecurityType.Equity, Market.USA)
        };
        AddUniverseSelection(new ManualUniverseSelectionModel(symbols));
        // Add another universe selection model to select the ones with EMA crosses.
        AddUniverseSelection(new EmaCrossUniverseSelectionModel());

        // Sent insights on buying and holding the most liquid stocks for a week.
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(7)));
        // Evenly dissipate the capital risk among selected stocks.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
Manual Selection - BasicTemplateFrameworkAlgorithm.cs C# Fundamental Selection - QC500UniverseSelectionModel.cs C#
Scheduled Selection - ScheduledUniverseSelectionModelRegressionAlgorithm.cs C#

Universe Selection

Universe Settings

Introduction

Universe settings and security initializers enable you to configure some properties of the securities in a universe.

Resolution

The `Resolution` setting defines the time period of the asset data. The `Resolution` enumeration has the following members:

To view which resolutions are available for the asset class of your universe, follow these steps:

1. Open the Asset Classes documentation page.
2. Click an asset class.
3. Click Requesting Data .
4. On the Requesting Data page, in the table of contents, click Resolutions .

The default value is `Resolution.Minute` . To change the resolution, in the Initialize method, adjust the algorithm's `UniverseSettings` before you create the Universe Selection model.

```
C#
// Set universe settings to subscribe to daily data and select SPY ETF constituents as the trading universe.
UniverseSettings.Resolution = Resolution.Daily;
AddUniverseSelection(new ETFConstituentsUniverseSelectionModel("SPY"));
```

Leverage

The `Leverage` setting is a `decimal` that defines the maximum amount of leverage you can use for a single asset in a non-derivative universe. The default value is `Security.NullLeverage` (0). To change the leverage, in the Initialize method, adjust the algorithm's `UniverseSettings` before you create the Universe Selection model.

```
C#
// Set unviverse leverage to 2x to increase potential returns from each asset in the non-derivative universe to
increase potential returns.
UniverseSettings.Leverage = 2.0m;
// Select universe based on EMA cross signals to identify and trade assets with trending behaviors.
AddUniverseSelection(new EmaCrossUniverseSelectionModel());
```

Fill Forward

The `FillForward` setting is a `bool` that defines whether or not too fill forward data. The default value is `true` . To disable fill forward in non-derivative universes, in the Initialize method, adjust the algorithm's `UniverseSettings` before you create the Universe Selection model.

```
// Disable fill forward to ensure each day's data is used exclusively, preventing previous day's data from
influencing decisions and select options for SPY with a daily update interval to track specific option contracts.
UniverseSettings.FillForward = false;
AddUniverseSelection(
    new OptionUniverseSelectionModel(
        TimeSpan.FromDays(1),
        _ => new [] { QuantConnect.Symbol.Create("SPY", SecurityType.Option, Market.USA) }
    )
);
```

Extended Market Hours

The `ExtendedMarketHours` setting is a `bool` that defines the trading schedule. If it's `true` , your algorithm receives price data for all trading hours. If it's `false` , your algorithm receives price data only for regular trading hours. The default value is `false` .

You only receive extended market hours data if you create the subscription with an intraday resolution. If you create the subscription with daily resolution, the daily bars only reflect the regular trading hours.

To view the trading schedule of an asset, open Asset Classes , click an asset class, then click Market Hours .

To enable extended market hours, in the Initialize method, adjust the algorithm's `UniverseSettings` before you create the Universe Selection model.

```
// Enable extended market hours for universe and manually select equity symbols for SPY, QQQ, and IWM to ensure
trading during pre-market and after-hours sessions.
UniverseSettings.ExtendedMarketHours = true;
var tickers = new[] {"SPY", "QQQ", "IWM"};
var symbols = tickers.Select(ticker => QuantConnect.Symbol.Create(ticker, SecurityType.Equity, Market.USA));
AddUniverseSelection(new ManualUniverseSelectionModel(symbols));
```

Minimum Time in Universe

The `MinimumTimeInUniverse` setting is a `TimeSpan` object that defines the minimum amount of time an asset must be in the universe before the universe can remove it. The default value is `TimeSpan.FromDays(1)` . To change the minimum time, in the Initialize method, adjust the algorithm's `UniverseSettings` before you create the Universe Selection model.

```
// Keep each security in the universe for a minimum of 7 days.
UniverseSettings.MinimumTimeInUniverse = TimeSpan.FromDays(7);
AddUniverseSelection(new ETFConstituentsUniverseSelectionModel("QQQ"));
```

Data Normalization Mode

The `DataNormalizationMode` setting is an enumeration that defines how historical data is adjusted. This setting is only applicable for US Equities and Futures.

In the case of US Equities, the data normalization mode affects how historical data is adjusted for corporate actions . To view all the available options, see Data Normalization . To change the data normalization mode, in the Initialize method, adjust the algorithm's `UniverseSettings` before you create the Universe Selection model.

In the case of Futures, the data normalization mode affects how historical data of two contracts is stitched together to form the continuous contract . To view all the available options, see Data Normalization .

The default value is `DataNormalizationMode.Adjusted` . To change the data normalization mode, in the Initialize method, adjust the algorithm's `UniverseSettings` before you create the Universe Selection model.

```
C#
// Set universe data normalization to use raw prices to reflect actual traded values without adjustments and manually
select equities MSTR, MSFT, and IBM.
UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;

var tickers = new[] {"MSTR", "MSFT", "IBM"};
var symbols = tickers.Select(ticker => QuantConnect.Symbol.Create(ticker, SecurityType.Equity, Market.USA));
AddUniverseSelection(new ManualUniverseSelectionModel(symbols));
```

Contract Depth Offset

The `ContractDepthOffset` setting is an `int` that defines which contract to use for the continuous Futures contract. 0 is the front month contract, 1 is the following back month contract, and 3 is the second back month contract. The default value is 0. To change the contract depth offset, in the Initialize method, adjust the algorithm's `UniverseSettings` before you create the Universe Selection model.

```
C#
// Select the following month contract dynamically to avoid low liquidity and high volatility, for S&P 500 E-mini
futures with daily updates.
UniverseSettings.ContractDepthOffset = 1;
AddUniverseSelection(
    new FutureUniverseSelectionModel(
        TimeSpan.FromDays(1),
        _ => new List<Symbol> {{ QuantConnect.Symbol.Create(Futures.Indices.SP500EMini, SecurityType.Future,
Market.CME) }}
    )
);
```

Asynchronous Selection

The `Asynchronous` setting is a `bool` that defines whether or not LEAN can run universe selection asynchronously, utilizing concurrent execution to increase the speed of your algorithm. The default value for this setting is `false` . If you enable this setting, abide by the following rules:

- Don't make any history requests in your filter function . History requests can only provide data up to the algorithm time , but if your filter function runs asynchronously, the algorithm time may not be what you expect.
- Don't use the portfolio, security, or orders state. The `Portfolio` , `Securities` , and `Transactions` objects are also functions of the algorithm time.
- If your filter function updates a class variable, don't update the class variable anywhere else in your algorithm.

To enable asynchronous universe selection, in the Initialize method, adjust the algorithm's `UniverseSettings` before you create the Universe Selection model.

```
C#
// Run universe selection asynchronously to speed up your algorithm.
UniverseSettings.Asynchronous = true;
AddUniverseSelection(new EmaCrossUniverseSelectionModel());
```

Schedule

The `Schedule` setting defines the selection schedule of the universe. Most universes run on a daily schedule. To change the selection schedule, call the `UniverseSettings.Schedule.On` method with an `IDateRule` object before you create the Universe Selection model.

```C#
// Trigger selections at the begining of each month.
UniverseSettings.Schedule.On(DateRules.MonthStart());
AddUniverseSelection(new ETFConstituentsUniverseSelectionModel("QQQ"));
```

The following table describes the supported `DateRules` :

Member	Description
`DateRules.SetDefaultTimeZone(DateTimeZone timeZone);`	Sets the time zone for the `DateRules` object used in all methods in this table. The default time zone is the algorithm time zone .
`DateRules.On(int year, int month, int day)`	Trigger an event on a specific date.
`DateRules.On(params DateTime[] dates)`	Trigger an event on specific dates.
`DateRules.EveryDay()`	Trigger an event all seven days of the week.
`DateRules.EveryDay(Symbol symbol, bool extendedMarketHours = false)`	Trigger an event every day a specific symbol is trading.
`DateRules.Every(params DayOfWeek[] days)`	Trigger an event on specific days throughout the week. To view the `DayOfWeek` enum members, see DayOfWeek Enum in the .NET documentation.
`DateRules.MonthStart(int daysOffset = 0)`	Trigger an event on the first day of each month plus an offset.
`DateRules.MonthStart(Symbol symbol, int daysOffset = 0)`	Trigger an event on the first tradable date of each month for a specific symbol plus an offset.
`DateRules.MonthEnd(int daysOffset = 0)`	Trigger an event on the last day of each month minus an offset.
`DateRules.MonthEnd(Symbol symbol, int daysOffset = 0)`	Trigger an event on the last tradable date of each month for a specific symbol minus an offset.
`DateRules.WeekStart(int daysOffset = 0)`	Trigger an event on the first day of each week plus an offset.
`DateRules.WeekStart(Symbol symbol, int daysOffset = 0)`	Trigger an event on the first tradable date of each week for a specific symbol plus an offset.
`DateRules.WeekEnd(int daysOffset = 0)`	Trigger an event on the last day of each week minus an offset.
`DateRules.WeekEnd(Symbol symbol, int daysOffset = 0)`	Trigger an event on the last tradable date of each week for a specific symbol minus an offset.
`DateRules.YearStart(int daysOffset = 0)`	Trigger an event on the first day of each year plus an offset.
`DateRules.YearStart(Symbol symbol, int daysOffset = 0)`	Trigger an event on the first tradable date of each year for a specific symbol plus an offset.
`DateRules.YearEnd(int daysOffset = 0)`	Trigger an event on the last day of each year minus an offset.
`DateRules.YearEnd(Symbol symbol, int daysOffset = 0)`	Trigger an event on the last tradable date of each year for a specific symbol minus an offset.
`DateRules.Today`	Trigger an event once today.
`DateRules.Tomorrow`	Trigger an event once tomorrow.

To define custom date rules, create a `FuncDateRule` object. The `FuncDateRule` constructor expects a `name` argument of type `string` and a `getDatesFunction` argument of type `Func<DateTime, DateTime, IEnumerable<DateTime>>` . The `getDatesFunction` function receives the start and end dates of the algorithm and returns a list of dates for the date rule. In live trading, the end date is 12/31/2025. The following example demonstrates how to define a date rule that represents the 10th day of each month:

```
C#

// Create a date rule that specifies the 10th day of each month.
var dateRule = new FuncDateRule(
    name: "10th_day_of_the_month",
    getDatesFunction: (start, end) => Enumerable.Range(start.Year, end.Year - start.Year + 1)
        .SelectMany(year => Enumerable.Range(1, 12).Select(month => new DateTime(year, month, 10)))
);
```

Some date rules require a Symbol. However, they accept a string representation of the Symbol as well. If your algorithm doesn't add the security , it assumes the string refers to a US Equity Symbol. For example, if you use the `DateRules.EveryDay("ES")` method, it assumes that 'ES' refers to a US Equity. To create a Symbol without adding a security to your algorithm, call the `QuantConnect.Symbol.Create` method.

```
C#

var symbol = QuantConnect.Symbol.Create("ES", SecurityType.Future, Market.CME);
```

In live trading, scheduled universes run at roughly 8 AM Eastern Time to ensure there is enough time for the data to process.

Configure Securities

Instead of configuring global universe settings, you can individually configure the settings of each security in the universe with a security initializer. Security initializers let you apply any security-level reality model or special data requests on a per-security basis. To set the security initializer, in the `Initialize` method, call the `SetSecurityInitializer` method and then define the security initializer.

```
C#

// A custom security initializer can override default models such as
// setting new fee and fill models for the security.
SetSecurityInitializer(CustomSecurityInitializer);

private void CustomSecurityInitializer(Security security)
{
    security.SetFeeModel(new ConstantFeeModel(0, "USD"));
}
```

For simple requests, you can use the functional implementation of the security initializer. This style lets you configure the security object with one line of code.

```
C#

// Disable the trading fees for each security by passing a functional
// implementation for the SetSecurityInitializer argument.
SetSecurityInitializer(security => security.SetFeeModel(new ConstantFeeModel(0, "USD")));
```

If you call the `SetSecurityInitializer` method, it overwrites the default security initializer. The default security initializer uses the security-level reality models of the brokerage model to set the following reality models of each security:

- Fill
- Slippage

- Fee
- Buying Power
- Settlement
- Short availability
- Margin Interest Rate

The default security initializer also sets the leverage of each security and intializes each security with a seeder function. To extend upon the default security initializer instead of overwriting it, call `AddSecurityInitializer` before you create the subscriptions.

```
C#
public class AddSecurityInitializerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the security initializer to set models of assets.
        AddSecurityInitializer(CustomSecurityInitializer);
    }

    private void CustomSecurityInitializer(Security security)
    {
        // Overwrite some of the reality models
        security.SetFeeModel(new ConstantFeeModel(0, "USD"));
    }
}
```

Examples

The following examples demonstrate some common practices for universe settings.

Example 1: Weekly-Updating Liquid Universe

The following algorithm demonstrates daily EMA cross, trading on the most liquid stocks. The universe is set to be updated weekly. Various universe settings have been set to simulate the brokerage environment best and for trading needs.

C#

```
public class UniverseSettingsAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // To avoid over-updating the universe, which causes insufficient time to earn the trend, update the universe
weekly.
        UniverseSettings.Schedule.On(DateRules.WeekStart());
        UniverseSettings.MinimumTimeInUniverse = TimeSpan.FromDays(7);
        // Rebalancing frequency is daily, so we only need to subscribe to daily resolution.
        UniverseSettings.Resolution = Resolution.Daily;
        // To best simulate IB's margin requirement set the leverage to 50%.
        UniverseSettings.Leverage = 2;
        // Since we trade by market order, we cannot use extended market hours data to fill.
        UniverseSettings.ExtendedMarketHours = false;
        // We want to trade the EMA with raw price but not altered by splits.
        UniverseSettings.DataNormalizationMode = DataNormalizationMode.SplitAdjusted;

                // Only trade on the top 10 most traded stocks since they have the most popularity to drive trends.
        AddUniverse(Universe.Top(10));
            }

    public override void OnData(Slice slice)
    {
        foreach (var (symbol, bar) in slice.Bars)
        {
            // Trade the trend by EMA crossing.
            var ema = (Securities[symbol] as dynamic).ema;
            if (bar.Close > ema)
            {
                SetHoldings(symbol, 0.05m);
            }
            else
            {
                SetHoldings(symbol, -0.05m);
            }
        }
    }

    public override void OnSecuritiesChanged(SecurityChanges changes)
    {
        foreach (var removed in changes.RemovedSecurities)
        {
            // Liquidate the ones leaving the universe since we will not trade their EMA anymore.
            Liquidate(removed.Symbol);
            // Deregister the EMA indicator to free resources
            DeregisterIndicator((removed as dynamic).ema as ExponentialMovingAverage);
        }

        foreach (var added in changes.AddedSecurities)
        {
            var security = added as dynamic;
            // Create EMA indicator for trend trading signals.
            security.ema = EMA(added.Symbol, 50, Resolution.Daily);
            // Warm up the EMA indicator to ensure its readiness for immediate use.
            WarmUpIndicator(added.Symbol, (ExponentialMovingAverage)security.ema, Resolution.Daily);
        }
    }
}
```

Universe Selection

Manual Universes

Introduction

The `ManualUniverseSelectionModel` selects a static, fixed set of assets. It is similar to adding securities with the traditional `AddSecurity` API methods. If your algorithm has a static universe, you can use automatic indicators instead of manual indicators in your algorithm.

Manual universes can be prone to look-ahead bias . For example, if you select a set of securities that have performed well during the backtest period, you are incorporating information from the future into the backtest and the algorithm may underperform in live mode.

Add Manual Universe Selection

To add a `ManualUniverseSelectionModel` to your algorithm, in the `Initialize` method, call the `AddUniverseSelection` method. The `ManualUniverseSelectionModel` constructor expects a list of `Symbol` objects that represent the universe constituents.

```
C#
// Use ManualUniverseSelectionModel method to select a static set of equities, similar to traditional add_security
methods, for a fixed trading universe.
var tickers = new[] {"SPY", "QQQ", "IWM"};
var symbols = tickers.Select(ticker => QuantConnect.Symbol.Create(ticker, SecurityType.Equity, Market.USA));
AddUniverseSelection(new ManualUniverseSelectionModel(symbols));
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`symbols`	`IEnumerable<Symbol>`	Universe constituents	
`universeSettings`	`UniverseSettings`	The universe settings . If you don't provide an argument, the model uses the `algorithm.UniverseSettings` by default.	

To move the universe tickers and `Symbol` objects outside of the algorithm class, create a universe selection model that inherits the `ManualUniverseSelectionModel` class.

```
// Create a custom universe selection model to move universe tickers and Symbol objects outside of the algorithm
class for improved modularity, reusability, and maintainability.
AddUniverseSelection(new IndexUniverseSelectionModel());

// Outside of the algorithm class
class IndexUniverseSelectionModel : ManualUniverseSelectionModel
{
    public IndexUniverseSelectionModel()
        : base(SelectSymbols()) {}

    public static IEnumerable<Symbol> SelectSymbols()
    {
        var tickers = new[] {"SPY", "QQQ", "IWM"};
        return tickers.Select(ticker => QuantConnect.Symbol.Create(ticker, SecurityType.Equity, Market.USA));
    }
}
```

C#

For more information about this model, see the class reference and implementation .

Examples

The following examples demonstrate some common practices for implementing a manual universe selection model.

Example 1: Crypto List Selection Model

The following algorithm selects a list of preset cryptos, given we have information that they will perform better. So, we can use `ManualUniverseSelectionModel` to select this list and buy the listed cryptos.

```
public class FrameworkManualUniverseSelectionAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Add a universe selection model to select the most liquid cryptos manually.
        var symbols = new [] {
            QuantConnect.Symbol.Create("BTCUSD", SecurityType.Crypto, Market.Coinbase),
            QuantConnect.Symbol.Create("ETHUSD", SecurityType.Crypto, Market.Coinbase)
        };
        AddUniverseSelection(new ManualUniverseSelectionModel(symbols));

        // Sent insights on buying and holding the most liquid cryptos for one week.
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(7)));
        // Evenly dissipate the capital risk among selected cryptos.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }
}
```

C#

Universe Selection

Fundamental Universes

Introduction

A `FundamentalUniverseSelectionModel` selects a universe of US Equities based on `Fundamental` data. Depending on the `Fundamental` properties you use, these universes rely on the US Equity Coarse Universe dataset, the US Fundamental dataset, or both.

These types of universes operate on daily schedule by default. In backtests, they select assets at midnight. In live trading, the selection timing depends on the data provider you use. To adjust the selection schedule, see Schedule .

If you use a fundamental Universe Selection model, the only way to unsubscribe from a security is to return a list from the selection function that doesn't include the security `Symbol` . The `RemoveSecurity` method doesn't work with these types of Universe Selection models.

Fundamental Selection

The `FundamentalUniverseSelectionModel` lets you select stocks based on corporate `Fundamental` data. You can specific the selection method, which takes a list of `Fundamental` objects as argument and returns a list of `Symbol` objects.

C#

```
public override void Initialize()
{
    // Run asynchronous universe selection to speed up your algorithm.
    // In this case, you can't rely on the method or algorithm state between filter calls.
    UniverseSettings.Asynchronous = true;
    AddUniverseSelection(new FundamentalUniverseSelectionModel(FundamentalFilterFunction));
}

public override List<Symbol> FundamentalFilterFunction(List<Fundamental> fundamental)
{
    return (from f in fundamental
            // Select symbols with fundamental data and a price above $10.
            where f.Price > 10 && !Double.IsNaN(f.ValuationRatios.PERatio)
            // Sort the assets in ascending order by their P/E ratio.
            orderby f.ValuationRatios.PERatio
            // Select the first 10 assets.
            select f.Symbol).Take(10);
}
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`selector`	`Func<IEnumerable<Fundamental>, IEnumerable<Symbol>>`	Filter function to select assets based on fundamental data.	
`universeSettings`	`UniverseSettings`	The universe settings . If you don't provide an argument, the model uses the `algorithm.UniverseSettings` by default.	`null`

The `Fundamental` objects have the following properties:

To move the selection function outside of the algorithm class, create a universe selection model that inherits the `FundamentalUniverseSelectionModel` class and override its `Select` method.

```
C#

// In the Initialize method, enable asynchronous universe selection to speed up your algorithm.
UniverseSettings.Asynchronous = true;
// Add a custom universe selection model that selects undervalued, liquid stocks.
AddUniverseSelection(new LiquidAndLowPEUniverseSelectionModel());

// Outside of the algorithm class, define the universe selection model.
public class LiquidAndLowPEUniverseSelectionModel : FundamentalUniverseSelectionModel
{
    public override IEnumerable<Symbol> Select(QCAlgorithm algorithm, IEnumerable<Fundamental> fundamental)
    {
        return fundamental
            // Select symbols with fundamental data and a price above $1.
            .Where(x => x.HasFundamentalData && x.Price > 1 && !Double.IsNaN(x.ValuationRatios.PERatio))
            // Select the 100 assets with the greatest daily dollar volume.
            .OrderByDescending(x => x.DollarVolume)
            .Take(100)
            // Select the 10 assets with the lowest PE ratio.
            .OrderByDescending(x => x.ValuationRatios.PERatio)
            .Take(10)
            .Select(x => x.Symbol);
    }
}
```

To return the current universe constituents from the selection function, return `Universe. Unchanged` .

For more information about this model, see the class reference and implementation .

EMA Cross Selection

The `EmaCrossUniverseSelectionModel` applies two exponential moving average (EMA) indicators to the price history of assets and then selects the assets that have their fast EMA furthest above their slow EMA on a percentage basis.

```
C#

// Initialize asynchronous universe settings for faster processing and add EmaCrossUniverseSelectionModel to
dynamically manage the universe based on EMA cross signals to identify trending assets.
public override void Initialize()
{
    UniverseSettings.Asynchronous = true;
    AddUniverseSelection(new EmaCrossUniverseSelectionModel());
}
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`fastPeriod`	`int`	Fast EMA period	100
`slowPeriod`	`int`	Slow EMA period	300
`universeCount`	`int`	Maximum number of members of this universe selection	500
`universeSettings`	`UniverseSettings`	The universe settings . If you don't provide an argument, the model uses the `algorithm.UniverseSettings` by default.	`null`

For more information about this model, see the class reference and implementation .

Examples

The following examples demonstrate some common practices for implementing the framework fundamental universe model.

Example 1: 50 Stocks >10/Share and >10M in Daily Trading Volume

The following algorithm selects the 50 most liquid US Equities above 10/share and 10M daily volume. We pass a function to the `FundamentalUniverseSelectionModel` to filter the fundamental data for stock selection.

C#

```
public class FrameworkFundamentalUniverseSelectionAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 8);
        SetCash(100000);

        // Configure the universe to update at the start of each month. Most of the top 500 don't change very
frequently.
        UniverseSettings.Schedule.On(DateRules.MonthStart());
        // Add a fundamental universe with custom selection rules for filtering.
        AddUniverseSelection(new FundamentalUniverseSelectionModel(fundamental =>
            (from f in fundamental
            //Large-cap non-penny stocks have more stable trends and demand.
            where f.Price > 10 && f.DollarVolume > 10000000
            orderby f.DollarVolume descending
            select f.Symbol).Take(500)
        ));

        // Sent insights on buying and holding the selected securities.
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(30)));
        // Evenly dissipate the capital risk among selected securities.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }
}
```

Example 2: 10 Stocks Above Their 200-Day EMA With >$1B of Daily Trading Volume

Another common request is to filter the universe by a technical indicator, such as only picking stocks above their 200-day Exponential Moving Average (EMA). The `Fundamental` object has adjusted price and volume information so that you can do any price-related analysis. The following algorithm defines a separate class to contain the indicator of each asset. We pass a function to the `FundamentalUniverseSelectionModel` to filter the fundamental data for stock selection.

```C#
using System.Collections.Concurrent;

public class FrameworkFrameworkUniverseSelectionAlgorithm : QCAlgorithm
{
    // Create a concurrent dictionary to store the EMA data for universe selection.
    private ConcurrentDictionary<Symbol, SelectionData> _selectionDataBySymbol = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Add a fundamental universe with custom selection rules for filtering.
        AddUniverseSelection(new FundamentalUniverseSelectionModel(Selection));

        // Sent insights on buying and holding the selected securities.
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(1)));
        // Evenly dissipate the capital risk among selected securities.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }

    private IEnumerable<Symbol> Selection(IEnumerable<Fundamental> fundamental)
    {
        return (from f in fundamental
            // Create/Update the EMA indicators of each stock.
            let avg = _selectionDataBySymbol.GetOrAdd(f.Symbol, sym => new SelectionData(this, f.Symbol, 200))
            where avg.Update(f.EndTime, f.AdjustedPrice)
            // Select the Equities above their EMA and have a daily volume of $1B.
            // These assets are in an uptrend and are very liquid.
            where avg.Ema.IsReady && f.Price > avg.Ema.Current.Value && f.DollarVolume > 1000000000
            // Select the 10 most liquid Equities to avoid extra slippage, and their trend can be capitalized more
efficiently.
            orderby f.DollarVolume descending
            select f.Symbol).Take(10);
    }
}

// Create a separate class to contain the EMA information of each asset.
public class SelectionData
{
    public readonly ExponentialMovingAverage Ema;

    // Create an EMA indicator for trend estimation and filtering.
    public SelectionData(QCAlgorithm algorithm, Symbol symbol, int period)
    {
        Ema = new ExponentialMovingAverage(period);

        // Warm up the EMA indicator.
        algorithm.WarmUpIndicator(symbol, Ema, Resolution.Daily);
    }

    // Update your variables and indicators with the latest data.
    // You may also want to use the History API here to warm up the indicator.
    public bool Update(DateTime time, decimal value)
    {
        return Ema.Update(time, value);
    }
}
```

In this example, the `SelectionData` class group variables for the universe selection and update the indicator of each asset. We highly recommend you follow this pattern to keep your algorithm tidy and bug-free. The following snippet shows an example implementation of the `SelectionData` class, but you can make whatever you need to store your custom universe filters. Note that the preceding `SelectionData` class uses a manual EMA indicator instead of the automatic version . For more information about universes that select assets based on indicators, see Indicator Universes . You need to use a `SelectionData` class instead of assigning the EMA to the `Fundamental` object because you can't create custom properties on `Fundamental` objects like you can with `Security` objects.

Example 3: 10 Stocks Furthest Above their 10-day SMA of Volume

The process to get the 10-day Simple Moving Average (SMA) stock volume is the same process as in Example 2. First, define a

`SelectionData` class that performs the averaging. This class tracks the ratio of today's volume relative to historical volumes. You can use this ratio to select assets above their 10-day SMA and sort the results by the Equities that have had the most significant jump since yesterday.

We inherit the `FundamentalUniverseSelectionModel` to create a custom universe selection model. Overriding the `Select` method, we filter the fundamental data for stocks selection.

```C#
using System.Collections.Concurrent;

public class FrameworkFrameworkUniverseSelectionAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);

        // Add a fundamental universe with custom selection rules for filtering.
        AddUniverseSelection(new VolumeSMAUniverseSelectionModel(10));

        // Sent insights on buying and holding the selected securities.
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(1)));
        // Evenly dissipate the capital risk among selected securities.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }
}

public class VolumeSMAUniverseSelectionModel : FundamentalUniverseSelectionModel
{
    // Create a concurrent dictionary to store the EMA data for universe selection.
    private ConcurrentDictionary<Symbol, SelectionData> _selectionDataBySymbol = new();
    private int _period;

    public VolumeSMAUniverseSelectionModel(int period)
        : base()
    {
        _period = period;
    }

    public override IEnumerable<Symbol> Select(QCAlgorithm algorithm, IEnumerable<Fundamental> fundamental)
    {
        return (from f in fundamental
            // Create/Update the volume SMA indicator of each stock.
            let avg = _selectionDataBySymbol.GetOrAdd(f.Symbol, sym => new SelectionData(algorithm, f.Symbol,
_period))
            where avg.Update(f.EndTime, f.Volume)
            // Select the Equities with higher trading volume than their SMA, indicating higher capital flow.
            where avg.VolumeRatio > 1
            // Select the 10 Equities with the highest relative volume since they have the highest capacity
            // for scalp-trading or intra-day movement.
            orderby avg.VolumeRatio descending
            select f.Symbol).Take(10);
    }

    // Define a separate class to contain and calculate the SMA of each Equity.
    private class SelectionData
    {
        public readonly Symbol Symbol;
        public readonly SimpleMovingAverage VolumeSma;
        public decimal VolumeRatio;

        public SelectionData(QCAlgorithm algorithm, Symbol symbol, int period)
        {
            // Create an SMA of volume to track the popularity of the stock.
            Symbol = symbol;
            VolumeSma = new SimpleMovingAverage(period);

            // Warm up the volume SMA indicator.
            algorithm.WarmUpIndicator(symbol, VolumeSma, Resolution.Daily, (data) => (data as TradeBar).Volume);
        }

        public bool Update(DateTime time, decimal value)
        {
            // Update the SMA with today's data and calculate the relative volume position for filtering.
            var ready = VolumeSma.Update(time, value);
```

```
            var sma = VolumeSma.Current.Value;
            if (sma > 0)
            {
                VolumeRatio = value / VolumeSma.Current.Value;
                return ready;
            }
            return false;
        }
    }
}
```

Example 4: 10 "Fastest Moving" Stocks With a 50-Day EMA > 200 Day EMA

You can construct complex universe filters with the `SelectionData` helper class pattern. To view a full example of this algorithm, see the EmaCrossUniverseSelectionAlgorithm in the LEAN GitHub repository or take the related Boot Camp lesson .

C#

```
using System.Collections.Concurrent;

public class FrameworkFrameworkUniverseSelectionAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Add a fundamental universe with custom selection rules for filtering.
        AddUniverseSelection(new EmaUniverseSelectionModel(10));

        // Sent insights on buying and holding the selected securities.
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(1)));
        // Evenly dissipate the capital risk among selected securities.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }
}

public class EmaUniverseSelectionModel : FundamentalUniverseSelectionModel
{
    // tolerance to prevent bouncing
    const decimal Tolerance = 0.01m;
    // Create a concurrent dictionary to store the indicator data for universe selection.
    private ConcurrentDictionary<Symbol, SelectionData> _averages = new();
    private int _selectionNumber;

    public EmaUniverseSelectionModel(int selectionNumber = 10)
        : base()
    {
        _selectionNumber = selectionNumber;
    }

    public override IEnumerable<Symbol> Select(QCAlgorithm algorithm, IEnumerable<Fundamental> fundamental)
    {
        return (from f in fundamental
            // Grab the SelectionData instance for this symbol
            let avg = _averages.GetOrAdd(f.Symbol, sym => new SelectionData(algorithm, f.Symbol))
            // Update returns true when the indicators are ready, so don't accept until they are
            where avg.Update(f.EndTime, f.AdjustedPrice)
            //Only pick symbols that have their 50-day ema over their 100-day ema
            where avg.Fast > avg.Slow * (1 + Tolerance)
            // prefer symbols with a larger delta by percentage between the two averages
            orderby avg.ScaledDelta descending
            // We only need to return the symbol and return selectionNumber symbols
            select f.Symbol).Take(_selectionNumber);
    }

    // class used to improve the readability of the coarse selection function
    private class SelectionData
    {
        public readonly ExponentialMovingAverage Fast;
        public readonly ExponentialMovingAverage Slow;

        public SelectionData(QCAlgorithm algorithm, Symbol symbol)
        {
            Fast = new ExponentialMovingAverage(100);
            Slow = new ExponentialMovingAverage(300);
```

```
            // Warm up the indicators.
            algorithm.WarmUpIndicator(symbol, Fast, Resolution.Daily);
            algorithm.WarmUpIndicator(symbol, Slow, Resolution.Daily);
        }

        // computes an object score of how large the fast is than the slow
        public decimal ScaledDelta
        {
            get { return (Fast - Slow)/((Fast + Slow)/2m); }
        }

        // updates the EMA50 and EMA100 indicators, returning true when they're both ready
        public bool Update(DateTime time, decimal value)
        {
            return Fast.Update(time, value) && Slow.Update(time, value);
        }
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
FundamentalUniverseSelectionAlgorithm.cs C# EmaCrossUniverseSelectionFrameworkAlgorithm.cs C#

Universe Selection

ETF Constituents Universes

Introduction

The `ETFConstituentsUniverseSelectionModel` selects a universe of US Equities based on the constituents of an ETF. These Universe Selection models rely on the US ETF Constituents dataset. They run on a daily schedule by default. To adjust the selection schedule, see Schedule .

Add ETF Constituents Universe Selection

To add an `ETFConstituentsUniverseSelectionModel` to your algorithm, in the `Initialize` method, call the `AddUniverseSelection` method. The `ETFConstituentsUniverseSelectionModel` constructor expects an ETF ticker.

```
C#
// Run universe selection asynchronously to speed up your algorithm.
UniverseSettings.Asynchronous = true;
AddUniverseSelection(new ETFConstituentsUniverseSelectionModel("SPY"));
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`etfTicker`	`string`	Ticker of the ETF to get constituents for. To view the available ETFs, see Supported ETFs .	
`universeSettings`	`UniverseSettings`	The universe settings . If you don't provide an argument, the model uses the `algorithm.UniverseSettings` by default.	
`universeFilterFunc`	`Func<IEnumerable<ETFConstituentUniverse>, IEnumerable<Symbol>>`	Function to filter ETF constituents. If you don't provide an argument, the model selects all of the ETF constituents by default.	`null`

If you provide a `universeFilterFunc` argument, you can use the following attributes of the `ETFConstituentUniverse` objects to select your universe:

The following example shows how to select the 10 Equities with the largest weight in the SPY ETF:

```C#
// Initialize asynchronous settings for speed and use the ETFConstituentsUniverseSelectionModel
// to select the top 10 SPY constituents by weight, focusing on blue-chip stocks with minimal risk.
public override void Initialize()
{
    UniverseSettings.Asynchronous = true;
    AddUniverseSelection(
        new ETFConstituentsUniverseSelectionModel("SPY", universeFilterFunc: ETFConstituentsFilter)
    );
}

private IEnumerable<Symbol> ETFConstituentsFilter(IEnumerable<ETFConstituentUniverse> constituents)
{
    // Select the 10 largest Equities in the ETF.
    return constituents.OrderByDescending(c => c.Weight).Take(10).Select(c => c.Symbol);
}
```

To move the ETF `Symbol` and the selection function outside of the algorithm class, create a universe selection model that inherits the `ETFConstituentsUniverseSelectionModel` class.

```C#
// Initialize asynchronous settings for speed and use the LargestWeightSPYETFUniverseSelectionModel
// to select the top 10 blue-chip SPY constituents by weight, focusing on stocks with minimal risk.
UniverseSettings.Asynchronous = true;
AddUniverseSelection(new LargestWeightSPYETFUniverseSelectionModel());

// Outside of the algorithm class
class LargestWeightSPYETFUniverseSelectionModel : ETFConstituentsUniverseSelectionModel
{
    public LargestWeightSPYETFUniverseSelectionModel(UniverseSettings universeSettings = null)
        : base("SPY", universeFilterFunc: ETFConstituentsFilter)
    {
    }

    private static IEnumerable<Symbol> ETFConstituentsFilter(IEnumerable<ETFConstituentUniverse> constituents)
    {
        // Select the 10 largest Equities in the ETF.
        return constituents.OrderByDescending(c => c.Weight).Take(10).Select(c => c.Symbol);
    }
}
```

To return the current universe constituents from the selection function, return `Universe. Unchanged` .

For more information about this model, see the class reference and implementation .

Examples

The following examples demonstrate some common practices for implementing the framework ETF constituent universe selection model.

Example 1: Weighted ETF Constituents

A subset of the SPY constituents outperform the SPY while many underperform the overall index. In an attempt to buy the ETF constituents that outperform the index, the following algorithm buys the top 50 weighted assets in the ETF. In this example, we will pass a function to the `ETFConstituentsUniverseSelectionModel` for selection.

C#

```
public class FrameworkETFConstituentsUniverseSelectionAlgorithm : QCAlgorithm
{
    public Dictionary<Symbol, double> EtfWeightBySymbol { get; set; } = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(10000000);

        // Add a universe of the SPY constituents.
        AddUniverseSelection(
            new ETFConstituentsUniverseSelectionModel("SPY", universeFilterFunc: ETFConstituentsFilter)
        );
        // Add an Alpha model to trade based on the constituent weights.
        AddAlpha(new EtfAlphaModel(this));
        // Position sizing was handled by insight weight (from ETF weight).
        SetPortfolioConstruction(new InsightWeightingPortfolioConstructionModel());
    }

    private IEnumerable<Symbol> ETFConstituentsFilter(IEnumerable<ETFConstituentUniverse> constituents)
    {
        // Cache the constituent weights in a dictionary for filtering and position sizing.
        EtfWeightBySymbol = constituents
            .Where(c => c.Weight.HasValue)
            .ToDictionary(c => c.Symbol, c => (double)c.Weight.Value);
        // Select the 50 constituents with the largest weight in the ETF.
        // They should have positive excess return.
        return EtfWeightBySymbol
            .OrderByDescending(x => x.Value)
            .Take(50)
            .Select(x => x.Key);
    }
}

public class EtfAlphaModel : AlphaModel
{
    private FrameworkETFConstituentsUniverseSelectionAlgorithm _algorithm;
    private static int _day = -1;

    public EtfAlphaModel(FrameworkETFConstituentsUniverseSelectionAlgorithm algorithm)
    {
        _algorithm = algorithm;
    }

    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
    {
        var insights = new List<Insight>();

        // Daily rebalance only.
        if (slice.Time.Day == _day)
        {
            return insights;
        }

        foreach (var (symbol, weight) in _algorithm.EtfWeightBySymbol)
        {
            // Get the ETF weight of all the assets currently in the universe.
            // To avoid trading errors, skip assets that have no price yet.
            if (slice.Bars.ContainsKey(symbol))
            {
                if (algorithm.Securities[symbol].Price == 0)
                {
                    continue;
                }
                insights.Add(Insight.Price(symbol, TimeSpan.FromDays(1), InsightDirection.Up, weight: weight));
            }
        }

        _day = slice.Time.Day;

        return insights;
    }
}
```

Example 2: Weight Trend Selection

The following algorithm trades the QQQ constituents with the upward trend of its weight constitution, indicated by an EMA indicator, suggesting its price trend is outperforming other constituents, or its weight was promoted by the NASDAQ index, which brings positive sentiment. To better utilize the EMA indicator, we inherit the `ETFConstituentsUniverseSelectionModel` superclass to create a custom universe selection model.

C#

```
public class FrameworkETFConstituentsUniverseSelectionAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(10000000);

        // Add a universe of the SPY constituents.
        AddUniverseSelection(
            new UptrendETFUniverseSelectionModel("QQQ")
        );
        // Add Alpha model to trade based on the selections.
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(1)));
        // Equally invest in insights to dissipate the capital risk evenly.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());

        // Warm up the indicators.
        SetWarmUp(60, Resolution.Daily);
    }
}

class UptrendETFUniverseSelectionModel : ETFConstituentsUniverseSelectionModel
{
    private static Dictionary<Symbol, ExponentialMovingAverage> _emaBySymbol = new();

    public UptrendETFUniverseSelectionModel(string etfTicker)
        : base(etfTicker, universeFilterFunc: ETFConstituentsFilter)
    {
    }

    private static IEnumerable<Symbol> ETFConstituentsFilter(IEnumerable<ETFConstituentUniverse> constituents)
    {
        // Remove the ones that are not in the ETF anymore.
        var toRemove = new HashSet<Symbol>(constituents.Select(x => x.Symbol).Except(_emaBySymbol.Keys));
        foreach (var symbol in toRemove)
        {
            _emaBySymbol.Remove(symbol);
        }

        foreach (var c in constituents)
        {
            if (_emaBySymbol.TryGetValue(c.Symbol, out var ema) && c.Weight.HasValue)
            {
                // Update EMA with the latest weight.
                ema.Update(c.EndTime, c.Weight.Value);
            }
            else
            {
                // Create an EMA to filter by trend.
                _emaBySymbol[c.Symbol] = new ExponentialMovingAverage(60);
            }
        }

        // Select the ones with the increasing trend of constituent weight.
        return _emaBySymbol.Where(kvp => kvp.Value.IsReady && kvp.Value > kvp.Value.Previous)
            .Select(kvp => kvp.Key);
    }
}
```

Universe Selection

Scheduled Universes

Introduction

A Scheduled Universe Selection model selects assets at fixed, regular intervals. In live trading, you can use this type of model to fetch tickers from Dropbox or to perform periodic historical analysis and select some assets.

Scheduled Universe Selection

The `ScheduledUniverseSelectionModel` selects assets on the schedule you provide. To use this model, provide a `DateRule` , `TimeRule` , and a selector function. The `DateRule` and `TimeRule` define the selection schedule. The selector function receives a `DateTime` object and returns a list of `Symbol` objects. The `Symbol` objects you return from the selector function are the constituents of the universe.

```C#
// Enable asynchronous universe selection to speed up your algorithm.
UniverseSettings.Asynchronous = true;
// Add a universe that selects assets at the beginning of each month.
AddUniverseSelection(
    new ScheduledUniverseSelectionModel(
        DateRules.MonthStart(),
        TimeRules.Midnight,
        // Select SPY for October. Otherwise, select QQQ.
        dt => new List<Symbol> {
            QuantConnect.Symbol.Create(dt.Month == 10 ? "SPY" : "QQQ", SecurityType.Equity, Market.USA)
        }
    )
);
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`dateRule`	`IDateRule`	Date rule defines what days the universe selection function will be invoked	
`timeRule`	`ITimeRule`	Time rule defines what times on each day selected by date rule the universe selection function will be invoked	
`selector`	`Func<DateTime, IEnumerable<Symbol>>`	Selector function accepting the date time firing time and returning the universe selected symbols	
`settings`	`UniverseSettings`	The universe settings . If you don't provide an argument, the model uses the `algorithm.UniverseSettings` by default.	`null`

The model assumes the `timeRule` argument is in Coordinated Universal Time (UTC). To use a different timezone, pass a `timeZone` argument of type `DateTimeZone` before the `dateRule` argument.

To return the current universe constituents from the selector function, return `Universe. Unchanged` .

```C#
// Add a universe that selects assets on Monday, Tuesday, and Thursday at 00:00, 06:00, 12:00, and 18:00 UTC.
AddUniverseSelection(new ScheduledUniverseSelectionModel(
    DateRules.Every(DayOfWeek.Monday, DayOfWeek.Tuesday, DayOfWeek.Thursday),
    TimeRules.Every(TimeSpan.FromHours(6)),
    SelectSymbols
));

// Define the selection function.
private IEnumerable<Symbol> SelectSymbols(DateTime dateTime)
{
    var tickers = new[] {"SPY", "AAPL", "IBM"};
    return tickers.Select(ticker =>
        QuantConnect.Symbol.Create(ticker, SecurityType.Equity, Market.USA));
}
```

For more information about this model, see the class reference and implementation .

Date Rules

The following table describes the supported `DateRules` :

Member	Description
`DateRules.SetDefaultTimeZone(DateTimeZone timeZone);`	Sets the time zone for the `DateRules` object used in all methods in this table. The default time zone is the algorithm time zone .
`DateRules.On(int year, int month, int day)`	Trigger an event on a specific date.
`DateRules.On(params DateTime[] dates)`	Trigger an event on specific dates.
`DateRules.EveryDay()`	Trigger an event all seven days of the week.
`DateRules.EveryDay(Symbol symbol, bool extendedMarketHours = false)`	Trigger an event every day a specific symbol is trading.
`DateRules.Every(params DayOfWeek[] days)`	Trigger an event on specific days throughout the week. To view the `DayOfWeek` enum members, see DayOfWeek Enum in the .NET documentation.
`DateRules.MonthStart(int daysOffset = 0)`	Trigger an event on the first day of each month plus an offset.
`DateRules.MonthStart(Symbol symbol, int daysOffset = 0)`	Trigger an event on the first tradable date of each month for a specific symbol plus an offset.
`DateRules.MonthEnd(int daysOffset = 0)`	Trigger an event on the last day of each month minus an offset.
`DateRules.MonthEnd(Symbol symbol, int daysOffset = 0)`	Trigger an event on the last tradable date of each month for a specific symbol minus an offset.
`DateRules.WeekStart(int daysOffset = 0)`	Trigger an event on the first day of each week plus an offset.
`DateRules.WeekStart(Symbol symbol, int daysOffset = 0)`	Trigger an event on the first tradable date of each week for a specific symbol plus an offset.
`DateRules.WeekEnd(int daysOffset = 0)`	Trigger an event on the last day of each week minus an offset.
`DateRules.WeekEnd(Symbol symbol, int daysOffset = 0)`	Trigger an event on the last tradable date of each week for a specific symbol minus an offset.
`DateRules.YearStart(int daysOffset = 0)`	Trigger an event on the first day of each year plus an offset.
`DateRules.YearStart(Symbol symbol, int daysOffset = 0)`	Trigger an event on the first tradable date of each year for a specific symbol plus an offset.
`DateRules.YearEnd(int daysOffset = 0)`	Trigger an event on the last day of each year minus an offset.
`DateRules.YearEnd(Symbol symbol, int daysOffset = 0)`	Trigger an event on the last tradable date of each year for a specific symbol minus an offset.
`DateRules.Today`	Trigger an event once today.
`DateRules.Tomorrow`	Trigger an event once tomorrow.

To define custom date rules, create a `FuncDateRule` object. The `FuncDateRule` constructor expects a `name` argument of type `string` and a `getDatesFunction` argument of type `Func<DateTime, DateTime, IEnumerable<DateTime>>` . The `getDatesFunction` function receives the start and end dates of the algorithm and returns a list of dates for the date rule. In live trading, the end date is 12/31/2025. The following example demonstrates how to define a date rule that represents the 10th day of each month:

```
C#
// Create a date rule that specifies the 10th day of each month.
var dateRule = new FuncDateRule(
    name: "10th_day_of_the_month",
    getDatesFunction: (start, end) => Enumerable.Range(start.Year, end.Year - start.Year + 1)
        .SelectMany(year => Enumerable.Range(1, 12).Select(month => new DateTime(year, month, 10)))
);
```

Some date rules require a Symbol. However, they accept a string representation of the Symbol as well. If your algorithm doesn't add the security , it assumes the string refers to a US Equity Symbol. For example, if you use the `DateRules.EveryDay("ES")` method, it assumes that 'ES' refers to a US Equity. To create a Symbol without adding a security to your algorithm, call the `QuantConnect.Symbol.Create` method.

```
C#
var symbol = QuantConnect.Symbol.Create("ES", SecurityType.Future, Market.CME);
```

Time Rules

The following table describes the supported `TimeRules` :

Member	Description
`TimeRules.SetDefaultTimeZone(DateTimeZone timeZone)`	Sets the time zone for the `TimeRules` object used in all methods in this table, except when a different time zone is given. The default time zone is the algorithm time zone .
`TimeRules.BeforeMarketOpen(Symbol symbol, double minutesBeforeOpen = 0, bool extendedMarketOpen = false)`	Trigger an event a few minutes before market open for a specific symbol (default is 0). This rule doesn't work for Crypto securities or custom data.
`TimeRules.AfterMarketOpen(Symbol symbol, double minutesAfterOpen = 0, bool extendedMarketOpen = false)`	Trigger an event a few minutes after market open for a specific symbol (default is 0). This rule doesn't work for Crypto securities or custom data.
`TimeRules.BeforeMarketClose(Symbol symbol, double minutesBeforeClose = 0, bool extendedMarketOpen = false)`	Trigger an event a few minutes before market close for a specific symbol (default is 0). This rule doesn't work for Crypto securities or custom data.
`TimeRules.AfterMarketClose(Symbol symbol, double minutesAfterClose = 0, bool extendedMarketOpen = false)`	Trigger an event a few minutes after market close for a specific symbol (default is 0). This rule doesn't work for Crypto securities or custom data.
`TimeRules.Every(TimeSpan interval)`	Trigger an event every period interval starting at midnight. This time rule triggers every period, regardless of whether or not the market is open.
`TimeRules.Now`	Trigger an event at the current time of day.
`TimeRules.Midnight`	Trigger an event at midnight.
`TimeRules.Noon`	Trigger an event at noon.
`TimeRules.At(int hour, int minute, int second = 0)`	Trigger an event at a specific time of day (e.g. 13:10).
`TimeRules.At(int hour, int minute, int second, DateTimeZone timeZone)`	Trigger an event at a specific time of day in the given time zone (e.g. 13:10 UTC).

To define custom time rules, create a `FuncTimeRule` object. The `FuncTimeRule` constructor expects a `name` argument of type `string` and a `createUtcEventTimesFunction` argument of type `Func<IEnumerable<DateTime>, IEnumerable<DateTime>>` . The function receives the list of dates from the date rule and then returns a list of `DateTime` that define the time rule.

```
C#
var timeRule = new FuncTimeRule(
    name: "CustomTimeRule",
    createUtcEventTimesFunction: dates => dates.Select(d => d.AddHours(10)));
```

Some time rules require a Symbol. However, they accept a string representation of the Symbol as well. If your algorithm doesn't add the security , it assumes the string refers to a US Equity Symbol. For example, if you use the `TimeRules.BeforeMarketOpen("ES", 15)` method, it assumes that 'ES' refers to a US Equity. To create a Symbol without adding a security to your algorithm, call the `QuantConnect.Symbol.Create` method.

```
C#
var symbol = QuantConnect.Symbol.Create("ES", SecurityType.Future, Market.CME);
```

Examples

The following examples demonstrate some common practices for implementing a dateless scheduled universe selection model.

Example 1: From External Source

The following example selects its universe every week starting before the market opens from a source URL of Dropbox. It reloads on every selection so that you can hook up to the continuously updating source file.

C#

```
public class FrameworkScheduledUniverseSelectionAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(10000000);

        // Add a universe of that read from a Dropbox source to select the stocks 30 minutes before the week's first
trading day.
        var spy = QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA);
        AddUniverseSelection(new ScheduledUniverseSelectionModel(
            DateRules.WeekStart(spy),
            TimeRules.BeforeMarketOpen(spy, 10),
            SelectSymbols)
        );
        // Add Alpha model to trade based on the selections; the signals last for a week until the next selection.
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(7)));
        // Equally invest in insights to dissipate the capital risk evenly.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }

    private string DownloadUniverseFile(DateTime dt)
    {
        // Download the universe CSV file. Dropbox links require the "dl=1" URL parameter.
        var file = Download(
            "https://www.dropbox.com/scl/fi/fbrxitk4ec3w91nse1raa/df.csv?
rlkey=7r042rukzkthp7y1srloyhkov&st=5r4sdfwd&dl=1"
        );
        // Convert the CSV file data into a dictionary where the key is the date and
        // the value is a comma-separated string of stock tickers.
        foreach (var line in file.Split('\n').Skip(1))
        {
            // Skip empty lines.
            if (line.IsNullOrEmpty())
            {
                continue;
            }
            var items = line.Split("\"");
            var date = Parse.DateTimeExact(items[0].Split(",")[0], "yyyy-MM-dd").Date;
            // Return the universe if the date matches.
            if (date == dt.Date)
            {
                return items[1];
            }
        }
        return string.Empty;
    }

    private IEnumerable<Symbol> SelectSymbols(DateTime dt)
    {
        // Re-download the CSV file each day to get today's row.
        var tickers = DownloadUniverseFile(dt);
        // If there isn't an entry for the current date, return an empty universe.
        if (tickers.IsNullOrEmpty())
        {
            return Enumerable.Empty<Symbol>();
        }
        // Convert the stock tickers in the CSV file to Symbol objects.
        return tickers
            .Split(',')
            .Select(x => QuantConnect.Symbol.Create(x, SecurityType.Equity, Market.USA));
    }
}
```

Example 2: Seasonality

The following algorithm selects between SPY and short-term bond ETF according to the current month. It holds SPY on May, June, July, November, and December since they are historically better-performing months while holding SHV to preserve funds and earn interest in the remaining months.

```C#
public class FrameworkScheduledUniverseSelectionAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);

        // Add a universe of that select based on the time to trade seasonality.
        AddUniverseSelection(new ScheduledUniverseSelectionModel(
            DateRules.WeekStart(),
            TimeRules.At(0, 0),
            SelectSymbols)
        );
        // Add an Alpha model to trade based on the selections; the signals will last until the next selection.
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(7)));
        // Equally invest in insights to dissipate the capital risk evenly.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }

    private IEnumerable<Symbol> SelectSymbols(DateTime dateTime)
    {
        // May/Jun/Jul/Nov/Dec are statistically better-performing months of SPY.
        if (new[] { 5, 6, 7, 11, 12 }.Contains(dateTime.Month))
        {
            return new[] { QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA) };
        }
        // In other months, hold short-term bonds to preserve funds while earning interest.
        return new[] { QuantConnect.Symbol.Create("SHV", SecurityType.Equity, Market.USA) };
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
ScheduledUniverseSelectionModelRegressionAlgorithm.cs C#

Universe Selection

Futures Universes

Introduction

A Future Universe Selection model selects contracts for a set of Futures.

Future Universe Selection

The `FutureUniverseSelectionModel` selects all the contracts for a set of Futures you specify. To use this model, provide a `refreshInterval` and a selector function. The `refreshInterval` defines how frequently LEAN calls the selector function. The selector function receives a `DateTime` object that represents the current Coordinated Universal Time (UTC) and returns a list of `Symbol` objects. The `Symbol` objects you return from the selector function are the Futures of the universe.

```
C#
// Run universe selection asynchronously to speed up your algorithm.
// This setting means you cannot rely on the method or algorithm state between filter calls.
UniverseSettings.Asynchronous = true;
// Add a universe of E-mini S&P 500 Futures contracts.
AddUniverseSelection(
    new FutureUniverseSelectionModel(
        // Refresh the universe daily.
        TimeSpan.FromDays(1),
        _ => new List<Symbol> {{ QuantConnect.Symbol.Create(Futures.Indices.SP500EMini, SecurityType.Future,
Market.CME) }}
    )
);
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`refreshInterval`	`TimeSpan`	Time interval between universe refreshes	
`futureChainSymbolSelector`	`Func<DateTime, IEnumerable<Symbol>>`	A function that selects the Future symbols for a given Coordinated Universal Time (UTC). To view the supported assets in the US Futures dataset, see Supported Assets .	
`universeSettings`	`UniverseSettings`	The universe settings . If you don't provide an argument, the model uses the `algorithm.UniverseSettings` by default.	`null`

The following example shows how to define the Future chain Symbol selector as an isolated method:

```C#
// In the Initialize method, add the FutureUniverseSelectionModel with a custom selection function.
public override void Initialize()
{
    AddUniverseSelection(
        new FutureUniverseSelectionModel(TimeSpan.FromDays(1), SelectFutureChainSymbols)
    );
}

private static IEnumerable<Symbol> SelectFutureChainSymbols(DateTime utcTime)
{
    // Add E-mini S&P 500 and Gold Futures to the universe.
    return new[] {
        QuantConnect.Symbol.Create(Futures.Indices.SP500EMini, SecurityType.Future, Market.CME),
        QuantConnect.Symbol.Create(Futures.Metals.Gold, SecurityType.Future, Market.COMEX)
    };
}
```

This model uses the default Future contract filter, which doesn't select any Futures contracts. To use a different filter, subclass the `FutureUniverseSelectionModel` and define a `Filter` method. The `Filter` method accepts and returns a `FutureFilterUniverse` object to select the Futures contracts. The following table describes the filter methods of the `FutureFilterUniverse` class:

`StandardsOnly()`	Selects standard contracts
`FrontMonth()`	Selects the front month contract
`BackMonths()`	Selects the non-front month contracts
`BackMonth()`	Selects the back month contracts
`Expiration(TimeSpan minExpiry, TimeSpan maxExpiry)`	Selects contracts that expire within a range of dates relative to the current day
`Expiration(int minExpiryDays, int maxExpiryDays)`	Selects contracts that expire within a range of dates relative to the current day
`Contracts(IEnumerable<Symbol> contracts)`	Selects a list of contracts
`Contracts(Func<IEnumerable<Symbol>, IEnumerable< Symbol>> contractSelector)`	Selects contracts that a selector function selects

The contract filter runs at the first time step of each day.

To move the Future chain Symbol selector and the contract selection function outside of the algorithm class, create a universe selection model that inherits the FundamentalUniverseSelectionModel class and override its Select method.

```
C#
// In the Initialize method, define the universe settings and add data.
UniverseSettings.Asynchronous = true;
AddUniverseSelection(new FrontMonthFutureUniverseSelectionModel());

// Outside of the algorithm class, define the universe selection model.
class FrontMonthFutureUniverseSelectionModel : FutureUniverseSelectionModel
{
    public FrontMonthFutureUniverseSelectionModel()
        // Refresh the universe daily.
        : base(TimeSpan.FromDays(1), SelectFutureChainSymbols) {}

    private static IEnumerable<Symbol> SelectFutureChainSymbols(DateTime utcTime)
    {
        // Add E-mini S&P 500 and Gold Futures to the universe.
        return new List<Symbol> {
            QuantConnect.Symbol.Create(Futures.Indices.SP500EMini, SecurityType.Future, Market.CME),
            QuantConnect.Symbol.Create(Futures.Metals.Gold, SecurityType.Future, Market.COMEX)
        };
    }

    protected override FutureFilterUniverse Filter(FutureFilterUniverse filter)
    {
        // Select the front month contracts.
        return filter.FrontMonth();
    }
}
```

Some of the preceding filter methods only set an internal enumeration in the `FutureFilterUniverse` that it uses later on in the filter process. This subset of filter methods don't immediately reduce the number of contract `Symbol` objects in the `FutureFilterUniverse` .

The `AddUniverseSelection` method doesn't return a `Future` object like the AddFuture method. The `Future` object contains `Symbol` and `Mapped` properties, which reference the continuous contract and the currently selected contract in the continuous contract series, respectively. To get the `Future` object, define the `OnSecuritiesChanged` method in your algorithm class or framework models and check the result of the `IsCanonical` method.

```
C#
// Save the Future object if the security added to the universe is the canonical asset.
public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
{
    foreach (var security in changes.AddedSecurities)
    {
        if (security.Symbol.IsCanonical() && security.Type == SecurityType.Future)
        {
            _future = security as Future;
        }
    }
}
```

For more information about this model, see the class reference and implementation .

Open Interest Future Universe Selection

The `OpenInterestFutureUniverseSelectionModel` is an extension of the `FutureUniverseSelectionModel` that selects the contract with the greatest open interest on a daily basis.

```
// Enable asynchronous universe settings for faster performance.
UniverseSettings.Asynchronous = true;
// Add an OpenInterestFutureUniverseSelectionModel for E-mini S&P 500 Futures, incorporating contracts with high
// open interest into the trading universe.
AddUniverseSelection(
    new OpenInterestFutureUniverseSelectionModel(
        this,
        utcTime => new[] { QuantConnect.Symbol.Create(Futures.Indices.SP500EMini, SecurityType.Future, Market.CME) }
    )
);
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`algorithm`	`IAlgorithm`	Algorithm	
`futureChainSymbolSelector`	`Func<DateTime, IEnumerable<Symbol>>`	A function that selects the Future symbols for a given Coordinated Universal Time (UTC). To view the supported assets in the US Futures dataset, see Supported Assets .	
`chainContractsLookupLimit`	`int?`	Limit on how many contracts to query for open interest	6
`resultsLimit`	`int?`	Limit on how many contracts will be part of the universe	1

The following example shows how to define the Future chain Symbol selector as an isolated method:

```
// In the Initialize method, define the universe settings and add a universe.
public override void Initialize()
{
    UniverseSettings.Asynchronous = true;
    AddUniverseSelection(
        new OpenInterestFutureUniverseSelectionModel(this, SelectFutureChainSymbols)
    );
}

// Define the selection function, which returns Symbol objects.
private static IEnumerable<Symbol> SelectFutureChainSymbols(DateTime utcTime)
{
    return new[] {
        QuantConnect.Symbol.Create(Futures.Indices.SP500EMini, SecurityType.Future, Market.CME),
        QuantConnect.Symbol.Create(Futures.Metals.Gold, SecurityType.Future, Market.COMEX)
    };
}
```

To move the Future chain Symbol selector outside of the algorithm class, create a universe selection model that inherits the `OpenInterestFutureUniverseSelectionModel` class.

```C#
// In the Initialize method, define the universe settings and add a universe.
UniverseSettings.Asynchronous = true;
AddUniverseSelection(new GoldOpenInterestFutureUniverseSelectionModel(this));

// Outside of the algorithm class, define the universe selection model.
class GoldOpenInterestFutureUniverseSelectionModel : OpenInterestFutureUniverseSelectionModel
{
    public GoldOpenInterestFutureUniverseSelectionModel(QCAlgorithm algorithm,
        int? chainContractsLookupLimit = 6, int? resultsLimit = 1)
        : base(algorithm, SelectFutureChainSymbols, chainContractsLookupLimit, resultsLimit) {}

    private static IEnumerable<Symbol> SelectFutureChainSymbols(DateTime utcTime)
    {
        return new List<Symbol> {
            QuantConnect.Symbol.Create(Futures.Metals.Gold, SecurityType.Future, Market.COMEX)
        };
    }
}
```

The `AddUniverseSelection` method doesn't return a `Future` object like the AddFuture method. The `Future` object contains `Symbol` and `Mapped` properties, which reference the continuous contract and the currently selected contract in the continuous contract series, respectively. To get the `Future` object, define the `OnSecuritiesChanged` method in your algorithm class or framework models and check the result of the `IsCanonical` method.

```C#
// Save the Future object if the security added to the universe is the canonical asset.
public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
{
    foreach (var security in changes.AddedSecurities)
    {
        if (security.Symbol.IsCanonical() && security.Type == SecurityType.Future)
        {
            _future = security as Future;
        }
    }
}
```

For more information about this model, see the class reference and implementation .

Examples

The following examples demonstrate some common practices for implementing a Futures universe selection model.

Example 1: Front Month Contracts

The following algorithm selects a list of Futures to buy and hold equally. To ensure liquidity and efficiency, we will select only their front-month contracts. We can do so in the algorithm framework through a child class inheriting the `FutureUniverseSelectionModel` while providing the list of Futures and overriding the `Filter` method to select the front month contracts.

```C#
public class FrameworkFutureUniverseSelectionAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // It is usual to trade Futures during extended market hours.
        UniverseSettings.ExtendedMarketHours = true;
        // Add a fundamental universe with custom selection rules for filtering.
        AddUniverseSelection(new FrontMonthFutureUniverseSelectionModel(10));

        // Sent insights on buying and holding the selected securities.
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(1)));
        // Evenly dissipate the capital risk among selected securities.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }

    private class FrontMonthFutureUniverseSelectionModel : FutureUniverseSelectionModel
    {
        /// Creates futures chain universes that select the front-month contract and run a user-defined
        /// futureChainSymbolSelector every day to enable choosing different futures chains
        public FrontMonthFutureUniverseSelectionModel(int rebalancePeriod)
            : base(TimeSpan.FromDays(rebalancePeriod), SelectFutureChainSymbols)
        {
        }

        protected override FutureFilterUniverse Filter(FutureFilterUniverse filter)
        {
            // Defines the futures chain universe filter to select only the front-month contracts at market open.
            return filter.FrontMonth();
        }

        private static IEnumerable<Symbol> SelectFutureChainSymbols(DateTime utcTime)
        {
            return [
                QuantConnect.Symbol.Create(Futures.Indices.VIX, SecurityType.Future, Market.CFE),
                QuantConnect.Symbol.Create(Futures.Indices.SP500EMini, SecurityType.Future, Market.CME),
                QuantConnect.Symbol.Create(Futures.Indices.NASDAQ100EMini, SecurityType.Future, Market.CME),
                QuantConnect.Symbol.Create(Futures.Indices.Dow30EMini, SecurityType.Future, Market.CBOT),
                QuantConnect.Symbol.Create(Futures.Energy.Gasoline, SecurityType.Future, Market.NYMEX),
                QuantConnect.Symbol.Create(Futures.Energy.HeatingOil, SecurityType.Future, Market.NYMEX),
                QuantConnect.Symbol.Create(Futures.Energy.NaturalGas, SecurityType.Future, Market.NYMEX),
                QuantConnect.Symbol.Create(Futures.Grains.Corn, SecurityType.Future, Market.CBOT),
                QuantConnect.Symbol.Create(Futures.Grains.Oats, SecurityType.Future, Market.CBOT),
                QuantConnect.Symbol.Create(Futures.Grains.Soybeans, SecurityType.Future, Market.CBOT),
                QuantConnect.Symbol.Create(Futures.Grains.Wheat, SecurityType.Future, Market.CBOT),
            ];
        }
    }
}
```

Example 2: Seasonal Contracts

Some Futures exhibit a seasonality effect, i.e., a yearly cycle in its price fluctuation. Studies show that the Natural Gas contract has a yearly cycle of maximizing its price in early December and minimizing it in early April. It might be due to the high demand for natural gas during winter warming. The following algorithm will hold Natural Gas contracts from summer to winter with a similar universe selection model in Example 1, but using a custom selection function with a time argument to select the contract.

C#

```
public class FrameworkFutureUniverseSelectionAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // It is usual to trade Futures during extended market hours.
        UniverseSettings.ExtendedMarketHours = true;
        // We hold the Future and will not frequently trade it; Daily resolution is sufficient for computational
efficiency.
        UniverseSettings.Resolution = Resolution.Daily;
        // Add a fundamental universe with custom selection rules for filtering.
        // Rescan every week to ensure we are on our plan.
        AddUniverseSelection(new FrontMonthFutureUniverseSelectionModel(7));

        // Sent insights on buying and holding the selected securities.
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(7)));
        // Place orders for a single contract to control risk exposure and avoid over-leveraging.
        SetPortfolioConstruction(new SingleSharePortfolioConstructionModel());
    }

    private class FrontMonthFutureUniverseSelectionModel : FutureUniverseSelectionModel
    {
        /// Creates futures chain universes that select the front-month contract and run a user-defined
        /// futureChainSymbolSelector every day to enable choosing different futures chains
        public FrontMonthFutureUniverseSelectionModel(int rebalancePeriod)
            : base(TimeSpan.FromDays(rebalancePeriod), SelectFutureChainSymbols)
        {
        }

        protected override FutureFilterUniverse Filter(FutureFilterUniverse filter)
        {
            // Defines the futures chain universe filter to select only the front-month contracts at market open.
            return filter.FrontMonth();
        }

        private static IEnumerable<Symbol> SelectFutureChainSymbols(DateTime utcTime)
        {
            // We hold Natural Gas from summer to winter since it is in an upward cycle.
            if (utcTime.Month >= 4 && utcTime.Month <= 11)
            {
                return [ QuantConnect.Symbol.Create(Futures.Energy.NaturalGas, SecurityType.Future, Market.NYMEX) ];
            }
            // We do not hold any contracts during the time of the downward cycle.
            return [];
        }
    }

    private class SingleSharePortfolioConstructionModel : PortfolioConstructionModel
    {
        public override IEnumerable<PortfolioTarget> CreateTargets(QCAlgorithm algorithm, Insight[] insights)
        {
            var targets = new List<PortfolioTarget>();
            foreach (var insight in insights)
            {
                if (algorithm.Securities[insight.Symbol].IsTradable)
                {
                    // Use integer target to create a portfolio target to trade a single contract
                    targets.Add(new PortfolioTarget(insight.Symbol, (int) insight.Direction));
                }
            }
            return targets;
        }
    }
}
```

Universe Selection

Options Universes

Introduction

An Option Universe Selection model selects contracts for a set of Options.

Options Universe Selection

The `OptionUniverseSelectionModel` selects all the available contracts for the Equity Options, Index Options, and Future Options you specify. To use this model, provide a `refreshInterval` and a selector function. The `refreshInterval` defines how frequently LEAN calls the selector function. The selector function receives a `DateTime` object that represents the current Coordinated Universal Time (UTC) and returns a list of `Symbol` objects. The `Symbol` objects you return from the selector function are the Options of the universe.

```
C#
// Run universe selection asynchronously to speed up your algorithm.
// In this case, you can't rely on the method or algorithm state between filter calls.
UniverseSettings.Asynchronous = true;
// Add a universe of SPY Options.
AddUniverseSelection(
    new OptionUniverseSelectionModel(
        // Refresh the universe daily.
        TimeSpan.FromDays(1),
        _ => new [] { QuantConnect.Symbol.Create("SPY", SecurityType.Option, Market.USA) }
    )
);
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`refreshInterval`	`TimeSpan`	Time interval between universe refreshes	
`optionChainSymbolSelector`	`Func<DateTime, IEnumerable<Symbol>>`	A function that selects the Option symbols	
`universeSettings`	`UniverseSettings`	The universe settings . If you don't provide an argument, the model uses the `algorithm.UniverseSettings` by default.	`null`

The following example shows how to define the Option chain Symbol selector as an isolated method:

```C#
// In the Initialize method, add the OptionUniverseSelectionModel with a custom selection function.
public override void Initialize()
{
    AddUniverseSelection(
        new OptionUniverseSelectionModel(TimeSpan.FromDays(1), SelectOptionChainSymbols)
    );
}

// Define the selection function.
private IEnumerable<Symbol> SelectOptionChainSymbols(DateTime utcTime)
{
    // Equity Options example:
    //var tickers = new[] {"SPY", "QQQ", "TLT"};
    //return tickers.Select(ticker => QuantConnect.Symbol.Create(ticker, SecurityType.Option, Market.USA));

    // Index Options example:
    //var tickers = new[] {"VIX", "SPX"};
    //return tickers.Select(ticker => QuantConnect.Symbol.Create(ticker, SecurityType.IndexOption, Market.USA));

    // Future Options example:
    var futureSymbol = QuantConnect.Symbol.Create(Futures.Indices.SP500EMini, SecurityType.Future, Market.CME);
    foreach (var contract in FuturesChain(futureSymbol))
    {
        yield return QuantConnect.Symbol.CreateCanonicalOption(contract.Symbol);
    }
}
```

This model uses the default Option filter, which selects all of the available Option contracts at the current time step. To use a different filter for the contracts, subclass the `OptionUniverseSelectionModel` and define a `Filter filter` method. The `Filter filter` method accepts and returns an `OptionFilterUniverse` object to select the Option contracts. The following table describes the methods of the `OptionFilterUniverse` class:

The following table describes the filter methods of the `OptionFilterUniverse` class:

`Strikes(int minStrike, int maxStrike)` Selects contracts that are within `minStrike` strikes below the underlying price and `maxStrike` strikes above the underlying price.
`CallsOnly()` Selects call contracts.
`PutsOnly()` Selects put contracts.
`StandardsOnly()` Selects standard contracts.
`IncludeWeeklys()` Selects non-standard weeklys contracts.
`WeeklysOnly()` Selects weekly contracts.
`FrontMonth()` Selects the front month contract.
`BackMonths()` Selects the non-front month contracts.
`BackMonth()` Selects the back month contracts.
`Expiration(int minExpiryDays, int maxExpiryDays)` Selects contracts that expire within a range of dates relative to the current day.
`Contracts(IEnumerable<Symbol> contracts)` Selects a list of contracts.
`Contracts(Func<IEnumerable<Symbol>, IEnumerable< Symbol>> contractSelector)` Selects contracts that a selector function selects.

The preceding methods return an `OptionFilterUniverse`, so you can chain the methods together.

The contract filter runs at the first time step of each day.

To move the Option chain Symbol selector outside of the algorithm class, create a universe selection model that inherits the `OptionUniverseSelectionModel` class.

```
C#
// In the Initialize method, define the universe settings and add data.
UniverseSettings.Asynchronous = true;
AddUniverseSelection(new EarliestExpiringAtTheMoneyCallOptionUniverseSelectionModel(this));

// Outside of the algorithm class, define the universe selection model.
class EarliestExpiringAtTheMoneyCallOptionUniverseSelectionModel : OptionUniverseSelectionModel
{
    public EarliestExpiringAtTheMoneyCallOptionUniverseSelectionModel(QCAlgorithm algorithm)
            : base(TimeSpan.FromDays(1), utcTime => SelectOptionChainSymbols(algorithm, utcTime)) {}

    private static IEnumerable<Symbol> SelectOptionChainSymbols(QCAlgorithm algorithm, DateTime utcTime)
    {
        // Equity Options example:
        //var tickers = new[] {"SPY", "QQQ", "TLT"};
        //return tickers.Select(ticker => QuantConnect.Symbol.Create(ticker, SecurityType.Option, Market.USA));

        // Index Options example:
        //var tickers = new[] {"VIX", "SPX"};
        //return tickers.Select(ticker => QuantConnect.Symbol.Create(ticker, SecurityType.IndexOption, Market.USA));

        // Future Options example:
        var futureSymbol = QuantConnect.Symbol.Create(Futures.Indices.SP500EMini, SecurityType.Future, Market.CME);
        foreach (var contract in algorithm.FuturesChain(futureSymbol))
        {
            yield return QuantConnect.Symbol.CreateCanonicalOption(contract.Symbol);
        }
    }

    // Create a filter to select contracts that have the strike price within 1 strike level and expire within 7
days.
    protected override OptionFilterUniverse Filter(OptionFilterUniverse filter)
    {
        return filter.Strikes(-1, -1).Expiration(0, 7).CallsOnly();
    }
}
```

Some of the preceding filter methods only set an internal enumeration in the `OptionFilterUniverse` that it uses later on in the filter process. This subset of filter methods don't immediately reduce the number of contract `Symbol` objects in the `OptionFilterUniverse`.

To override the default pricing model of the Options, set a pricing model in a security initializer.

To override the initial guess of implied volatility, set and warm up the underlying volatility model.

For more information about this model, see the class reference and implementation.

Option Chained Universe Selection

An Option chained universe subscribes to Option contracts on the constituents of a US Equity universe.

```
// Configure the universe to use price data unadjusted for splits and dividends ("raw") into the algorithm.
// Options require raw Equity prices.
UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;
UniverseSettings.Asynchronous = true;
AddUniverseSelection(
    new OptionChainedUniverseSelectionModel(
        // Add a universe of the 10 most liquid US Equities.
        AddUniverse(Universe.Top(10)),
        // Select call Option contracts on the underlying Equities that have the strike price within 2 strike levels.
        optionFilterUniverse => optionFilterUniverse.Strikes(-2, +2).FrontMonth().CallsOnly()
    )
);
```

C#

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`universe`	`Universe`	The universe to chain onto the Option Universe Selection model	
`optionFilter`	`Func<OptionFilterUniverse, OptionFilterUniverse>`	The Option filter universe to use	
`universeSettings`	`UniverseSettings`	The universe settings . If you don't provide an argument, the model uses the `algorithm.UniverseSettings` by default.	`null`

The `optionFilter` function receives and returns an `OptionFilterUniverse` to select the Option contracts. The following table describes the methods of the `OptionFilterUniverse` class:

The following table describes the filter methods of the `OptionFilterUniverse` class:

`Strikes(int minStrike, int maxStrike)` Selects contracts that are within `minStrike` strikes below the underlying price and `maxStrike` strikes above the underlying price.
`CallsOnly()` Selects call contracts.
`PutsOnly()` Selects put contracts.
`StandardsOnly()` Selects standard contracts.

`IncludeWeeklys()` Selects non-standard weeklys contracts.
`WeeklysOnly()` Selects weekly contracts.
`FrontMonth()` Selects the front month contract.
`BackMonths()` Selects the non-front month contracts.
`BackMonth()` Selects the back month contracts.
`Expiration(int minExpiryDays, int maxExpiryDays)` Selects contracts that expire within a range of dates relative to the current day.
`Contracts(IEnumerable<Symbol> contracts)` Selects a list of contracts.
`Contracts(Func<IEnumerable<Symbol>, IEnumerable< Symbol>> contractSelector)` Selects contracts that a selector function selects.

The preceding methods return an `OptionFilterUniverse` , so you can chain the methods together.

The following example shows how to define the Option filter as an isolated method:

```
C#
```

```
// In the Initialize method, define the universe settings and add the universe selection model.
public override void Initialize()
{
    UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;
    UniverseSettings.Asynchronous = true;
    AddUniverseSelection(
        new OptionChainedUniverseSelectionModel(
            AddUniverse(Universe.Top(10)), OptionFilterFunction
        )
    );
}

// Define the contract filter function to select front month call contracts with a strike price within 2 strike
levels.
private OptionFilterUniverse OptionFilterFunction(OptionFilterUniverse optionFilterUniverse)
{
    return optionFilterUniverse.Strikes(-2, +2).FrontMonth().CallsOnly();
}
```

Some of the preceding filter methods only set an internal enumeration in the `OptionFilterUniverse` that it uses later on in the filter process. This subset of filter methods don't immediately reduce the number of contract `Symbol` objects in the `OptionFilterUniverse` .

For more information about this model, see the class reference and implementation .

Examples

The following examples demonstrate some common practices for implementing the framework Option Universe Selection Model.

Example 1: Horizontal Jelly Roll

The following algorithm selects SPX index options to construct a Jelly Roll strategy. It filters for ATM calls and puts with 30 days and 90 days till expiration. Using the SMA indicator to predict the interest rate cycle, it longs Jelly Roll if the cycle is considered uprising, otherwise selling the Jelly Roll.

```C#
public class FrameworkOptionUniverseSelectionAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(500000);

        // Add a universe that selects the needed option contracts.
        AddUniverseSelection(new AtmOptionHorizontalSpreadUniverseSelectionModel());
        // Add Alpha model to trade Jelly Roll, using interest rate data.
        AddAlpha(new JellyRollAlphaModel(this));
        // Invest in the same number of contracts per leg in the Jelly Roll.
        SetPortfolioConstruction(new SingleSharePortfolioConstructionModel());
    }

    private class AtmOptionHorizontalSpreadUniverseSelectionModel : OptionUniverseSelectionModel
    {
        // 30d update with the SelectOptionChainSymbols function since the filter returns at least 30d expiry
options.
        public AtmOptionHorizontalSpreadUniverseSelectionModel()
                : base(TimeSpan.FromDays(30), SelectOptionChainSymbols) {}

        private static IEnumerable<Symbol> SelectOptionChainSymbols(DateTime utcTime)
        {
            // We will focus only on SPX options since they have a relatively stable dividend yield, which we assume
will remain the same over time.
            // Also, assignment handling is not required since it is cash-settled.
            return new[] {QuantConnect.Symbol.Create("SPX", SecurityType.IndexOption, Market.USA)};
        }

        protected override OptionFilterUniverse Filter(OptionFilterUniverse filter)
        {
            // To trade interest rates using options, Jelly Roll is one of the best strategies.
            // It is market-neutral but sensitive to interest rate and dividend yield changes.
            // We target to trade the market speculation between 30d and 90d options interest rate.
            return filter.JellyRoll(0, 30, 90);
        }
    }

    private class JellyRollAlphaModel : AlphaModel
    {
        private QCAlgorithm _algorithm;
        private Symbol _symbol = QuantConnect.Symbol.Create("SPX", SecurityType.IndexOption, Market.USA);
        private bool _wasRising;
        // Use a 365d SMA indicator of daily interest rate to estimate if the interest rate cycle is upward or
downward.
        private SimpleMovingAverage _sma = new(365);

        public JellyRollAlphaModel(QCAlgorithm algorithm)
        {
            _algorithm = algorithm;

            // Warm up the SMA indicator.
            var current = algorithm.Time;
```

```
            var current = algorithm.Time;
            var provider = algorithm.RiskFreeInterestRateModel;
            for (DateTime dt = current.AddDays(-365); dt <= current; dt = dt.AddDays(1))
            {
                var rate = provider.GetInterestRate(dt);
                _sma.Update(dt, rate);
                _wasRising = rate > _sma;
            }

            // Set a schedule to update the interest rate trend indicator daily.
            algorithm.Schedule.On(
                algorithm.DateRules.EveryDay(),
                algorithm.TimeRules.At(0, 1),
                UpdateInterestRate
            );
        }

        private void UpdateInterestRate()
        {
            // Update the interest rate on the SMA indicator to estimate its trend.
            var rate = _algorithm.RiskFreeInterestRateModel.GetInterestRate(_algorithm.Time);
            _sma.Update(_algorithm.Time, rate);
            _wasRising = rate > _sma;
        }

        public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
        {
            var insights = new List<Insight>();

            // Hold one position group at a time.
            if (algorithm.Portfolio.Invested || !slice.OptionChains.TryGetValue(_symbol, out var chain))
            {
                return insights;
            }

            // Obtain the Jelly Roll constituents from the option chain.
            var calls = chain.Where(x => x.Right == OptionRight.Call)
                .OrderBy(x => x.Expiry)
                .ToList();
            var puts = chain.Where(x => x.Right == OptionRight.Put)
                .OrderBy(x => x.Expiry)
                .ToList();
            var nearCall = calls[0];
            var farCall = calls[^1];
            var nearPut = puts[0];
            var farPut = puts[^1];

            // Emit insight of the Jelly Roll constituents, with directions depending on the interest rate trend
given by SMA.
            var rate = algorithm.RiskFreeInterestRateModel.GetInterestRate(algorithm.Time);
            List<Insight> insightGroup;
            // During the rising interest rate cycle, order a long Jelly Roll.
            if (rate > _sma)
            {
                insightGroup = new() {
                    Insight.Price(nearCall.Symbol, TimeSpan.FromDays(30), InsightDirection.Down),
                    Insight.Price(farCall.Symbol, TimeSpan.FromDays(30), InsightDirection.Up),
                    Insight.Price(nearPut.Symbol, TimeSpan.FromDays(30), InsightDirection.Up),
                    Insight.Price(farPut.Symbol, TimeSpan.FromDays(30), InsightDirection.Down)
                };
            }
            // During a downward interest rate cycle, order short Jelly Roll.
            else if (rate < _sma)
            {
                insightGroup = new() {
                    Insight.Price(nearCall.Symbol, TimeSpan.FromDays(30), InsightDirection.Up),
                    Insight.Price(farCall.Symbol, TimeSpan.FromDays(30), InsightDirection.Down),
                    Insight.Price(nearPut.Symbol, TimeSpan.FromDays(30), InsightDirection.Down),
                    Insight.Price(farPut.Symbol, TimeSpan.FromDays(30), InsightDirection.Up)
                };
            }
            // If the interest rate cycle remains steady for a long time, we expect a flip in the cycle soon.
            else if (_wasRising)
            {
                insightGroup = new() {
                    Insight.Price(nearCall.Symbol, TimeSpan.FromDays(30), InsightDirection.Up),
                    Insight.Price(farCall.Symbol, TimeSpan.FromDays(30), InsightDirection.Down),
                    Insight.Price(nearPut.Symbol, TimeSpan.FromDays(30), InsightDirection.Down),
                    Insight.Price(farPut.Symbol, TimeSpan.FromDays(30), InsightDirection.Up)
                };
            }
            else
```

```
            else
            {
                insightGroup = new() {
                    Insight.Price(nearCall.Symbol, TimeSpan.FromDays(30), InsightDirection.Down),
                    Insight.Price(farCall.Symbol, TimeSpan.FromDays(30), InsightDirection.Up),
                    Insight.Price(nearPut.Symbol, TimeSpan.FromDays(30), InsightDirection.Up),
                    Insight.Price(farPut.Symbol, TimeSpan.FromDays(30), InsightDirection.Down)
                };
            }
            insights.AddRange(insightGroup);

            return insights;
        }
    }

    private class SingleSharePortfolioConstructionModel : PortfolioConstructionModel
    {
        public override IEnumerable<PortfolioTarget> CreateTargets(QCAlgorithm algorithm, Insight[] insights)
        {
            var targets = new List<PortfolioTarget>();
            foreach (var insight in insights)
            {
                if (algorithm.Securities[insight.Symbol].IsTradable)
                {
                    // Use integer target to create a portfolio target to trade a single contract
                    targets.Add(new PortfolioTarget(insight.Symbol, (int) insight.Direction));
                }
            }
            return targets;
        }
    }
}
```

Example 2: Options of an Equity Universe

The following example chains a fundamental universe and an Equity Options universe . It first selects 10 stocks with the lowest PE ratio and then selects their front-month call Option contracts. It buys one front-month call Option contract every day.

To override the default pricing model of the Options, set a pricing model in a security initializer.

To override the initial guess of implied volatility , set and warm up the underlying volatility model .

C#

```
// Example code to chain a fundamental universe and an Equity Options universe by selecting top 10 stocks with lowest
PE, indicating potentially undervalued stocks and then selecting their from-month call Option contracts to target
contracts with high liquidity.
public class ETFUniverseOptions : QCAlgorithm
{
    private int _day;
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
        Settings.SeedInitialPrices = true;
        UniverseSettings.Asynchronous = true;
        UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;
        AddSecurityInitializer(security =>
        {
            if (security.Type == SecurityType.Option) // Option type
            {
                (security as Option).PriceModel = OptionPriceModels.CrankNicolsonFD();
            }

            // Overwrite the volatility model and warm it up
            if (security.Type == SecurityType.Equity)
            {
                security.VolatilityModel = new StandardDeviationOfReturnsVolatilityModel(30);
                var tradeBars = History(security.Symbol, 30, Resolution.Daily);
                foreach (var tradeBar in tradeBars)
                    security.VolatilityModel.Update(security, tradeBar);
            }
        });

        var universe = AddUniverse(FundamentalFunction);
        AddUniverseOptions(universe, OptionFilterFunction);
    }

    private IEnumerable<Symbol> FundamentalFunction(IEnumerable<Fundamental> fundamental)
    {
        return fundamental
            .Where(f => !double.IsNaN(f.ValuationRatios.PERatio))
            .OrderBy(f => f.ValuationRatios.PERatio)
            .Take(10)
            .Select(x => x.Symbol);
    }

    private OptionFilterUniverse OptionFilterFunction(OptionFilterUniverse optionFilterUniverse)
    {
        return optionFilterUniverse.Strikes(-2, +2).FrontMonth().CallsOnly();
    }

    public override void OnData(Slice data)
    {
        if (IsWarmingUp || _day == Time.Day) return;

        foreach (var (symbol, chain) in data.OptionChains)
        {
            if (Portfolio[chain.Underlying.Symbol].Invested)
                Liquidate(chain.Underlying.Symbol);

            var spot = chain.Underlying.Price;
            var contract = chain.OrderBy(x => Math.Abs(spot-x.Strike)).FirstOrDefault();
            var tag = $"IV: {contract.ImpliedVolatility:F3} Δ: {contract.Greeks.Delta:F3}";
            MarketOrder(contract.Symbol, 1, true, tag);
            _day = Time.Day;
        }
    }
}
```

Universe Selection

Legacy Fundamental Universes

Introduction

Warning: This API for Universe Selection was deprecated on November 2023. Please refer to the new Fundamental Selection API .

A `FundamentalUniverseSelectionModel` selects a universe of US Equities based on `CoarseFundamental` and, sometimes, `FineFundamental` data. If the model uses `CoarseFundamental` data, it relies on the US Equity Coarse Universe dataset. If the Universe Selection model uses `FineFundamental` data, it relies on the US Fundamental dataset.

These types of universes operate on daily schedule. In backtests, they select assets at midnight. In live trading, the selection timing depends on the data provider you use.

If you use a fundamental Universe Selection model, the only way to unsubscribe from a security is to return a list from the selection functions that doesn't include the security `Symbol` . The `RemoveSecurity` method doesn't work with these types of Universe Selection models.

Coarse Fundamental Selection

The `CoarseFundamentalUniverseSelectionModel` selects assets based on `CoarseFundamental` data. To use this model, define a coarse selection function. The coarse selection function receives a list of `CoarseFundamental` objects and returns a list of `Symbol` objects. The `Symbol` objects you return from the coarse selection function are the constituents of the universe.

```C#
// Set asynchronous universe settings for improved speed and use CoarseFundamentalUniverseSelectionModel with a
custom function to select the top 100 most liquid assets with available fundamental data, ensuring the universe
consists of liquid and fundamentally strong securities for stable returns.
public override void Initialize()
{
    UniverseSettings.Asynchronous = true;
    AddUniverseSelection(new CoarseFundamentalUniverseSelectionModel(SelectCoarse));
}

private IEnumerable<Symbol> SelectCoarse(IEnumerable<CoarseFundamental> coarse)
{
    // Return most liquid assets w/ fundamentals
    return coarse.Where(c => c.HasFundamentalData)
                 .OrderByDescending(c => c.DollarVolume)
                 .Take(100)
                 .Select(c => c.Symbol);
}
```

To move the coarse selection function outside of the algorithm class, create a universe selection model that inherits the `CoarseFundamentalUniverseSelectionModel` class.

```C#
// Move the coarse selection function outside of the algorithm class by creating a custom universe selection model
inheriting from CoarseFundamentalUniverseSelectionModel, allowing for modification and reusability of the selection
logic.
UniverseSettings.Asynchronous = true;
AddUniverseSelection(new MostLiquidFundamentalUniverseSelectionModel(UniverseSettings));

// Outside of the algorithm class
class MostLiquidFundamentalUniverseSelectionModel : CoarseFundamentalUniverseSelectionModel
{
    public MostLiquidFundamentalUniverseSelectionModel(UniverseSettings universeSettings)
        : base(default(Func<IEnumerable<CoarseFundamental>, IEnumerable<Symbol>>), universeSettings)
    {
    }

    public override IEnumerable<Symbol> SelectCoarse(QCAlgorithm algorithm, IEnumerable<CoarseFundamental> coarse)
    {
        return coarse.Where(c => c.HasFundamentalData)
                     .OrderByDescending(c => c.DollarVolume)
                     .Take(100)
                     .Select(c => c.Symbol);
    }
}
```

To return the current universe constituents from the coarse selection function, return `Universe.Unchanged` .

For more information about this model, see the class reference and implementation .

Fine Fundamental Selection

The `FineFundamentalUniverseSelectionModel` selects assets based on `CoarseFundamental` and `FineFundamental` data. This is the only model that provides corporate fundamental data to your algorithm. To use this model, define a coarse selection function and a fine selection function. The coarse selection function receives a list of `CoarseFundamental` objects and returns a list of `Symbol` objects. To filter the `CoarseFundamental` down to the securities that have fundamental data, add a `HasFundamentalData` filter to the coarse selection function. The fine selection function receives a subset of `FineFundamental` objects generated from coarse selection results and returns a list of `Symbol` objects. The `Symbol` objects you return from the fine selection function are the constituents of the universe.

```C#
// FineFundamentalUniverseSelectionModel integrates corporate fundamental data into the algorithm through a two-step
filtering process: coarse selection ensures liquidity and fundamental data availability, while fine selection focuses
on valuation metrics, refining the universe of selected stocks.
public override void Initialize()
{
    UniverseSettings.Asynchronous = true;
    AddUniverseSelection(new FineFundamentalUniverseSelectionModel(SelectCoarse, SelectFine));
}

private IEnumerable<Symbol> SelectCoarse(IEnumerable<CoarseFundamental> coarse)
{
    // Return most liquid assets w/ fundamentals
    return coarse.Where(c => c.HasFundamentalData)
                 .OrderByDescending(c => c.DollarVolume)
                 .Take(100)
                 .Select(c => c.Symbol);
}

private IEnumerable<Symbol> SelectFine(IEnumerable<FineFundamental> fine)
{
    // Return assets with lowest P/E ratios
    return fine.OrderBy(f => f.ValuationRatios.PERatio)
               .Take(10)
               .Select(f => f.Symbol);
}
```

To move the selection functions outside of the algorithm class, create a universe selection model that inherits the `FineFundamentalUniverseSelectionModel` class.

C#

```
// Create a custom universe selection model inheriting from FineFundamentalUniverseSelectionModel to move both coarse
and fine selection functions outside the algorithm class, enabling easier modification and reusability of the
filtering and ranking logic.
UniverseSettings.Asynchronous = true;
AddUniverseSelection(new LowPERatioUniverseSelectionModel(UniverseSettings));

// Outside of the algorithm class
class LowPERatioUniverseSelectionModel : FineFundamentalUniverseSelectionModel
{
    public LowPERatioUniverseSelectionModel(UniverseSettings universeSettings = null)
        : base(default(Func<IEnumerable<CoarseFundamental>, IEnumerable<Symbol>>),
               default(Func<IEnumerable<FineFundamental>, IEnumerable<Symbol>>),
               universeSettings)
    {
    }

    public override IEnumerable<Symbol> SelectCoarse(QCAlgorithm algorithm, IEnumerable<CoarseFundamental> coarse)
    {
        return coarse.Where(c => c.HasFundamentalData)
                     .OrderByDescending(c => c.DollarVolume)
                     .Take(100)
                     .Select(c => c.Symbol);
    }

    public override IEnumerable<Symbol> SelectFine(QCAlgorithm algorithm, IEnumerable<FineFundamental> fine)
    {
        return fine.OrderBy(f => f.ValuationRatios.PERatio)
                   .Take(10)
                   .Select(f => f.Symbol);
    }
}
```

To return the current universe constituents from the coarse or fine selection function, return `Universe. Unchanged` .

If you add a `FineFundamentalUniverseSelectionModel` to your algorithm, you can access fundamental data in the fine selection function or from the `Equity` object. To access fundamental data from the `Equity` object, use the `Equity. Fundamentals` property.

For more information about this model, see the class reference and implementation .

Algorithm Framework

Alpha

Alpha

Key Concepts

Introduction

The Alpha model predicts market trends and signals the best moments to trade. These signals, or `Insight` objects, contain the `Direction` , `Magnitude` , and `Confidence` of a market prediction and the suggested portfolio `Weight` . You should generate insights on the set of assets provided by the Universe Selection model and only generate them when your predictions change.

Add Models

To add an Alpha model, in the `Initialize` method, call the `AddAlpha` method.

```
C#
// Add EmaCrossAlphaModel to generate market trend insights and trading signals based on EMA crossovers, providing
Direction, Magnitude, and Confidence for the universe-selected assets.
AddAlpha(new EmaCrossAlphaModel());
```

To view all the pre-built Alpha models, see Supported Models .

Multi-Alpha Algorithms

You can add multiple Alpha models to a single algorithm and generate Insight objects with all of them.

```
C#
// Add RsiAlphaModel and EmaCrossAlphaModel to generate trading signals based on RSI and EMA crossovers, utilizing
multiple alpha models to capture different market signals and trends for robust predictions.
AddAlpha(new RsiAlphaModel());
AddAlpha(new EmaCrossAlphaModel());
```

Each Alpha model has a unique name. The Insight objects they generate are automatically named according to the Alpha model name.

If you add multiple Alpha models, each alpha model receives the current slice in the order that you add the Alphas. The combined stream of Insight objects is passed to the Portfolio Construction model that defines how the Insight collection is combined. If you have a hybrid algorithm, the combined stream of Insight objects is passed to the event handler .

Model Structure

Alpha models should extend the `AlphaModel` class. Extensions of the `AlphaModel` class must implement the `Update` method, which receives a Slice object and returns an array of `Insight` objects. Extensions should also implement the `OnSecuritiesChanged` method to track security changes in the universe.

```C#
// Algorithm framework model that produces insights
class MyAlphaModel : AlphaModel
{
    // Updates this Alpha model with the latest data from the algorithm.
    // This is called each time the algorithm receives data for subscribed securities
    public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice data)
    {
        var insights = new List<Insight>();
        return insights;
    }

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        // Security additions and removals are pushed here.
        // This can be used for setting up algorithm state.
        // changes.AddedSecurities
        // changes.RemovedSecurities
    }
}
```

Method Parameters

The `algorithm` parameter that the methods receive is an instance of the base `QCAlgorithm` class, not your subclass of it. To access members of your algorithm object, either use a global variable or pass an algorithm reference to Alpha model constructor. Both of these approaches violate the separation of concerns principle.

The `data` parameter contains the latest data available to the algorithm.

The `changes` parameter contains the universe changes.

Model Names

To ensure your Alpha model is compatible with all Portfolio Construction models , assign a unique name to your Alpha model. Some Portfolio Construction models can combine multiple Alpha models together, so it can be important to distinguish between the Alpha models.

```C#
// Assign a unique name to RsiAlphaModel to ensure compatibility and distinguish it when combining with other Alpha
models in portfolio construction.
public class RsiAlphaModel : AlphaModel
{
    // Give your alpha a name (perhaps based on its constructor args?)
    public override string Name { get; }
}
```

By default, LEAN uses the string representation of a Guid object to set the Alpha model name. An example default name is "0f8fad5b-d9cb-469f-a165-70867728950e". This default name is different for every backtest you run and every live algorithm you deploy. For consistent behavior, manually set the Alpha model name.

Example

To view a full example of an `AlphaModel` subclass, see the ConstantAlphaModel in the LEAN GitHub repository.

Track Security Changes

The Universe Selection model may select a dynamic universe of assets, so you should not assume a fixed set of assets in the Alpha model. When the Universe Selection model adds and removes assets from the universe, it triggers an `OnSecuritiesChanged` event. In the `OnSecuritiesChanged` event handler, you can initialize the security-specific state or load any history required for your Alpha model. If you need to save data for individual securities, cast the `Security` object to a

`dynamic` object and then save custom members to it .

```C#
// Handle dynamic asset changes by initializing indicators for new securities and removing them for removed
securities to keep the Alpha model adaptive and accurate.
class MyAlphaModel : AlphaModel
{
    private List<Security> _securities = new List<Security>();

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes) {
        foreach (var security in changes.AddedSecurities)
        {
            var dynamicSecurity = security as dynamic;
            dynamicSecurity.Sma = algorithm.SMA(security.Symbol, 20);
            algorithm.WarmUpIndicator(security.Symbol, dynamicSecurity.Sma);
            _securities.Add(security);
        }

        foreach (var security in changes.RemovedSecurities)
        {
            if (_securities.Contains(security))
            {
                algorithm.DeregisterIndicator((security as dynamic).Sma);
                _securities.Remove(security);
            }
        }
    }
}
```

Insights

An Insight is a single prediction for an asset. The `Update` method returns an array of Insight objects. You can think of these as actionable trading signals, indicating the asset direction, magnitude, and confidence in the near future. All insights can take a weight parameter to set the desired weighting for the insight. If you change the cash of the algorithm, it will affect the orders, but not necessarily the insights.

The Portfolio Construction model consumes the Insight objects from the Alpha model. It's up to the Portfolio Construction model to define how Insight objects are converted into `PortfolioTarget` objects. In the pre-built Portfolio Construction models, down insights translate to short-biased trades, up insights translate to long-biased trades, and flat insights usually close open positions, but this doesn't have to be the case in custom Portfolio Construction models .

Insight Properties

`Insight` objects have the following properties:

Create Insights

To create an `Insight` , call the `Insight` constructor.

```C#
// new Insight(symbol, period, type, direction, magnitude=null, confidence=null, sourceModel=null, weight=null,
tag="");
var insight = new Insight("IBM", TimeSpan.FromMinutes(20), InsightType.Price, InsightDirection.Up);
```

In the `Insight` constructor, the `period` argument can be a `TimeSpan` object or a function that receives a `DateTime` object and returns the expiry `DateTime` . Some of the constructor arguments are optional to create the `Insight` object, but some of the Portfolio Construction models may require these properties. To check which `Insight` properties a Portfolio Construction model requires, see Supported Models .

You can also use the helper method to create Insight objects of the Price type.

```C#
// new Insight(symbol, period, direction, magnitude=null, confidence=null, sourceModel=null, weight=null, tag="")
var insight = Insight.Price("IBM", TimeSpan.FromMinutes(20), InsightDirection.Up);

// new Insight(symbol, resolution, barCount, direction, magnitude=null, confidence=null, sourceModel=null,
weight=null, tag="")
var insight = Insight.Price("IBM", Resolution.Minute, 20, InsightDirection.Up);
```

In the `Price` method, the `period` argument can be a `TimeSpan` object, a `DateTime` object, or a function that receives a `DateTime` object and returns the expiry `DateTime` .

Group Insights

Sometimes an algorithm's performance relies on multiple insights being traded together, like in pairs trading and options straddles. These types insights should be grouped. Insight groups signal to the Execution model that the insights need to be acted on as a single unit to maximize the alpha created.

To mark insights as a group, call the `Insight.Group` method.

```C#
// Generate and group insights for a set of tickers with a 20-minute prediction horizon; each Insight object provides
directional signals, magnitudes, and confidence for price movement.
return Insight.Group(
 Insight.price("IBM", TimeSpan.FromMinutes(20), InsightDirection.UP),
 Insight.price("SPY", TimeSpan.FromMinutes(20), InsightDirection.UP),
 Insight.price("AAPL", TimeSpan.FromMinutes(20), InsightDirection.UP)
);
```

Cancel Insights

If an Alpha model in your algorithm emits an Insight to enter a position but it determines the trading opportunity has pre-maturely ended, you should cancel the Insight. For example, say you want your algorithm to enter a long position in a security when its Relative Strength Index (RSI) moves below 20 and then liquidate the position when the security's RSI moves above 30. If you emit an Insight that has a duration of 30 days when the RSI moves below 20 but the RSI moves above 30 in 10 days, you should cancel the Insight.

To cancel an Insight, call its `Cancel` / `Expire` method with the algorithm's Coordinated Universal Time (UTC).

```C#
// Cancel any outstanding insights to ensure only current predictions are considered and to avoid outdated signals
influencing trading decisions.
_insight.Cancel(algorithm.UtcTime);
```

If you don't have a reference to the Insight you want to cancel, get it from the InsightManager .

When you cancel an insight, it's `CloseTimeUtc` property is set to one second into the past.

Stop Loss Orders Workaround

In some cases, if you add a Risk Management model that uses stop loss logic, the Risk Management model generates `PortfolioTarget` objects with a 0 quantity to make the Execution model liquidate your positions, but then the Portfolio Construction model generates `PortfolioTarget` objects with a non-zero quantity to make the Execution model re-enter the same position. This issue can occur if your Portfolio Construction model rebalances and your Alpha model still has an active insight for the liquidated securities. To avoid this issue, add the stop loss order logic to the Alpha model. When the stop loss is hit, emit a flat insight for the security. Note that this is a workaround, but it violates the separation of concerns principle since the Alpha Model shouldn't react to open positions.

Universe Timing Considerations

If the Alpha model manages some indicators or consolidators for securities in the universe and the universe selection runs during the indicator sampling period or the consolidator aggregation period, the indicators and consolidators might be missing some data. For example, take the following scenario:

- The security resolution is minute
- You have a consolidator that aggregates the security data into daily bars to update the indicator
- The universe selection runs at noon

In this scenario, you create and warm-up the indicator at noon. Since it runs at noon, the history request that gathers daily data to warm up the indicator won't contain any data from the current day and the consolidator that updates the indicator also won't aggregate any data from before noon. This process doesn't cause issues if the indicator only uses the close price to calculate the indicator value (like the simple moving average indicator) because the first consolidated bar that updates the indicator will have the correct close price. However, if the indicator uses more than just the close price to calculate its value (like the True Range indicator), the open, high, and low values of the first consolidated bar may be incorrect, causing the initial indicator values to be incorrect.

Examples

The following examples demonstrate common practices for implementing the framework alpha model.

Example 1: EMA Cross On Liquid Stocks

The following algorithm selects the top 500 liquid US stocks traded within the SPY ETF, then emits trade signals 60-day versus 200-day EMA cross through the `EmaCrossAlphaModel`. The `EqualWeightingPortfolioConstructionModel` ensures each asset with an insight gets an allocation of equal magnitude.

```
C#
public class FrameworkAlphaModelAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Add a universe of liquid equities since they have capital to support demand and trends.
        UniverseSettings.Resolution = Resolution.Daily;
        AddUniverseSelection(new ETFConstituentsUniverseSelectionModel("SPY"));
        // Using an EMA cross-alpha model to trade medium versus long-term trends.
        AddAlpha(new EmaCrossAlphaModel(60, 200, Resolution.Daily));
        // Equally invested in insights to dissipate capital risk evenly.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }
}
```

Example 2: Interest Rate Effect On Forex

The following algorithm implements a custom alpha model that uses the interest rate cycle as a signal to trade Forex. We trade the top 6 forexes: AUDUSD, EURUSD, GBPUSD, USDCAD, USDCHF, and USDJPY. For the ones with USD as quote currencies, we buy them during the US interest rate down cycle since USD is relatively weak while shorting them during the upcycle. The opposite is true for the ones with USD as the base currency.

```
C#
public class FrameworkAlphaModelAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
```

```
        SetEndDate(2024, 12, 31);

        // Use daily resolution since the interest rate signal is on a daily basis.
        UniverseSettings.Resolution = Resolution.Daily;
        // Add a universe of selected lists of Forex.
        var forex = new[] {
            QuantConnect.Symbol.Create("AUDUSD", SecurityType.Forex, Market.Oanda),
            QuantConnect.Symbol.Create("EURUSD", SecurityType.Forex, Market.Oanda),
            QuantConnect.Symbol.Create("GBPUSD", SecurityType.Forex, Market.Oanda),
            QuantConnect.Symbol.Create("USDCAD", SecurityType.Forex, Market.Oanda),
            QuantConnect.Symbol.Create("USDCHF", SecurityType.Forex, Market.Oanda),
            QuantConnect.Symbol.Create("USDJPY", SecurityType.Forex, Market.Oanda)
        };
        AddUniverseSelection(new ManualUniverseSelectionModel(forex));
        // Using a custom alpha model to trade forex based on interest rate.
        AddAlpha(new InterestRateAlphaModel(this));
        // Equally invested in insights to dissipate capital risk evenly.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }

    private class InterestRateAlphaModel : AlphaModel
    {
        private QCAlgorithm _algorithm;
        private bool _wasRising;
        // Use a 365d SMA indicator of daily interest rate to estimate if the interest rate cycle is upward or
downward.
        private SimpleMovingAverage _sma = new(365);

        public InterestRateAlphaModel(QCAlgorithm algorithm)
        {
            _algorithm = algorithm;

            // Warm up the SMA indicator.
            var current = algorithm.Time;
            var provider = algorithm.RiskFreeInterestRateModel;
            for (DateTime dt = current.AddDays(-365); dt <= current; dt = dt.AddDays(1))
            {
                var rate = provider.GetInterestRate(dt);
                _sma.Update(dt, rate);
                _wasRising = rate > _sma;
            }

            // Set schedule to update the interest rate trend indicator daily.
            algorithm.Schedule.On(
                algorithm.DateRules.EveryDay(),
                algorithm.TimeRules.At(0, 1),
                UpdateInterestRate
            );
        }

        private void UpdateInterestRate()
        {
            // Update the interest rate on the SMA indicator to estimate its trend.
            var rate = _algorithm.RiskFreeInterestRateModel.GetInterestRate(_algorithm.Time);
            _sma.Update(_algorithm.Time, rate);
            _wasRising = rate > _sma;
        }

        public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice slice)
        {
            var insights = new List<Insight>();

            // Split the forexes by whether the quote currency is USD.
            var quoteUSD = new List<Symbol>();
            var baseUSD = new List<Symbol>();
            foreach (var (symbol, security) in algorithm.Securities)
            {
                if (security.QuoteCurrency.Symbol == Currencies.USD)
                {
                    quoteUSD.Add(symbol);
                }
                else
                {
                    baseUSD.Add(symbol);
                }
            }

            var rate = algorithm.RiskFreeInterestRateModel.GetInterestRate(algorithm.Time);
            // During the rising interest rate cycle, long the forexes with USD as the base currency and short the
ones with USD as the quote currency.
            if (rate > _sma)
```

```
            {
                insights.AddRange(
                    baseUSD.Select(symbol => Insight.Price(symbol, TimeSpan.FromDays(1),
InsightDirection.Up)).ToList()
                );
                insights.AddRange(
                    quoteUSD.Select(symbol => Insight.Price(symbol, TimeSpan.FromDays(1),
InsightDirection.Down)).ToList()
                );
            }
            // During a downward interest rate cycle, short the forexes with USD as the base currency and long the
ones with USD as the quote currency.
            else if (rate < _sma)
            {
                insights.AddRange(
                    baseUSD.Select(symbol => Insight.Price(symbol, TimeSpan.FromDays(1),
InsightDirection.Down)).ToList()
                );
                insights.AddRange(
                    quoteUSD.Select(symbol => Insight.Price(symbol, TimeSpan.FromDays(1),
InsightDirection.Up)).ToList()
                );
            }
            // If the interest rate cycle remains steady for a long time, we expect a flip in the cycle soon.
            else if (_wasRising)
            {
                insights.AddRange(
                    baseUSD.Select(symbol => Insight.Price(symbol, TimeSpan.FromDays(1),
InsightDirection.Down)).ToList()
                );
                insights.AddRange(
                    quoteUSD.Select(symbol => Insight.Price(symbol, TimeSpan.FromDays(1),
InsightDirection.Up)).ToList()
                );
            }
            else
            {
                insights.AddRange(
                    baseUSD.Select(symbol => Insight.Price(symbol, TimeSpan.FromDays(1),
InsightDirection.Up)).ToList()
                );
                insights.AddRange(
                    quoteUSD.Select(symbol => Insight.Price(symbol, TimeSpan.FromDays(1),
InsightDirection.Down)).ToList()
                );
            }

            return insights;
        }
    }
}
```

Alpha

Supported Models

Introduction

This page describes the pre-built Alpha models in LEAN. The number of models grows over time. To add a model to LEAN, make a pull request to the GitHub repository . If none of these models perform exactly how you want, create a custom Alpha model .

Null Model

The `NullAlphaModel` doesn't emit any insights. It's the default Alpha model.

```
C#
// NullAlphaModel doesn't emit insights.
AddAlpha(new NullAlphaModel());
```

For more information about this model, see the class reference and implementation .

Constant Model

The `ConstantAlphaModel` always returns the same insight for each security.

```
C#
// Emit up insights that expire in 30 days for all assets.
AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(30)));
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`type`	`InsightType`	The type of insight	
`direction`	`InsightDirection`	The direction of the insight	
`period`	`TimeSpan`	The period over which the insight will come to fruition	
`magnitude`	`double?`	The predicted change in magnitude as a +/- percentage	`null`
`confidence`	`double?`	The confidence in the insight	`null`
`weight`	`double?`	The portfolio weight of the insights	`null`

For more information about this model, see the class reference and implementation .

Historical Returns Model

The `HistoricalReturnsAlphaModel` buys securities that have a positive trailing return and sells securities that have a negative

trailing return. It sets the magnitude of the Insight objects to the trailing rate of change.

```
// Add HistoricalReturnsAlphaModel to leverage historical return data for generating alpha signals, identifying
trends based on past performance.
AddAlpha(new HistoricalReturnsAlphaModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`lookback`	`int`	Historical return lookback period	1
`resolution`	`Resolution`	The resolution of historical data	`Resolution.Daily`

This model cancels all the active insights it has emit for a security when the security has a 0% historical return.

For more information about this model, see the class reference and implementation .

EMA Cross Model

The `EmaCrossAlphaModel` uses an exponential moving average (EMA) cross to create insights. When the fast EMA crosses above the slow EMA, it emits up insights. When the fast EMA crosses below the slow EMA, it emits down insights. It sets the duration of Insight objects to be the product of the `resolution` and `fastPeriod` arguments.

```
// Add EmaCrossAlphaModel to generate trading insights based on the crossing of fast and slow exponential moving
averages identifying trends and potential reversals. The duration of insights is set by the resolution and
fast_period to ensure timely and relevant trading signals.
AddAlpha(new EmaCrossAlphaModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`fastPeriod`	`int`	The fast EMA period	12
`slowPeriod`	`int`	The slow EMA period	26
`resolution`	`Resolution`	The resolution of data sent into the EMA indicators	`Resolution.Daily`

For more information about this model, see the class reference and implementation .

MACD Model

The `MacdAlphaModel` emits insights based on moving average convergence divergence (MACD) crossovers. If the MACD signal line is 1% above the security price, the model emits an up insight. If the MACD signal line is 1% below the security price, the model emits a down insight. If the MACD signal line is within 1% of the security price, the model cancels all the active insights it has emitted for the security.

```
// Use MacdAlphaModel to generate insights based on MACD crossovers, emitting up signals when the MACD signal line is
1% above the security price and down signals when it is 1% below, while canceling insights if within 1% capturing
significant trend changes while avoiding noise from minor fluctuations.
AddAlpha(new MacdAlphaModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`fastPeriod`	`int`	The MACD fast period	12
`slowPeriod`	`int`	The MACD slow period	26
`signalPeriod`	`int`	The smoothing period for the MACD signal	9
`movingAverageType`	`MovingAverageType`	The type of moving average to use in the MACD	`MovingAverageType. Exponential`
`resolution`	`Resolution`	The resolution of data sent into the MACD indicator	`Resolution.Daily`

For more information about this model, see the class reference and implementation .

RSI Model

The `RsiAlphaModel` generates insights based on the relative strength index (RSI) indicator values. When the RSI value passes above 70, the model emits a down insight. When the RSI value passes below 30, the model emits an up insight. The model uses the Wilder moving average type and sets the duration of Insight objects to be the product of the `resolution` and `period` arguments.

```
// The RsiAlphaModel generates insights based on the RSI indicator: emits a down insight when RSI exceeds 70
(overbought) and an up insight when RSI falls below 30 (oversold). This helps identify potential price reversals
based on momentum.
AddAlpha(new RsiAlphaModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`period`	`int`	The RSI indicator period	14
`resolution`	`Resolution`	The resolution of data sent into the RSI indicator	`Resolution.Daily`

For more information about this model, see the class reference and implementation .

Base Pairs Trading Model

The `BasePairsTradingAlphaModel` analyzes every possible pair combination from securities that the Universe Selection model selects. This model calculates a ratio between the two securities by dividing their historical prices over a lookback window. It then calculates the mean of this ratio by taking the 500-period EMA of the quotient. When the ratio diverges far enough from the mean ratio, this model emits generates alternating long ratio/short ratio insights emitted as a group to capture the reversion

of the ratio.

```C#
// Use BasePairsTradingAlphaModel to analyze security pairs from the universe, generating long/short insights based on deviations from a 500-period EMA of their price ratio to capture mean-reversion opportunities.
AddAlpha(new BasePairsTradingAlphaModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`lookback`	`int`	Lookback period of the analysis	1
`resolution`	`Resolution`	Analysis resolution	`Resolution.Daily`
`threshold`	`decimal`	The percent [0, 100] deviation of the ratio from the mean before emitting an insight	1

For more information about this model, see the class reference and implementation .

Pearson Correlation Pairs Trading Model

The `PearsonCorrelationPairsTradingAlphaModel` ranks every pair combination by its Pearson correlation coefficient and trades the pair with the highest correlation. This model follows the same insight logic as the `BasePairsTradingModel` .

```C#
// Add PearsonCorrelationPairsTradingAlphaModel to rank and trade the most correlated security pairs based on Pearson correlation coefficients, using a similar insight logic as BasePairsTradingModel to capture mean-reversion opportunities.
AddAlpha(new PearsonCorrelationPairsTradingAlphaModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`lookback`	`int`	Lookback period of the analysis	15
`resolution`	`Resolution`	Analysis resolution	`Resolution.Minute`
`threshold`	`decimal`	The percent [0, 100] deviation of the ratio from the mean before emitting an insight	1
`minimumCorrelation`	`double`	The minimum correlation to consider a tradable pair	0.5

For more information about this model, see the class reference and implementation .

Algorithm Framework

Portfolio Construction

Portfolio Construction

Key Concepts

Introduction

The Portfolio Construction model receives `Insight` objects from the Alpha model and creates `PortfolioTarget` objects for the Risk Management model. A `PortfolioTarget` provides the number of units of an asset to hold.

Set Models

To set a Portfolio Construction model, in the `Initialize` method, call the `SetPortfolioConstruction` method.

```
C#
// Create PortfolioTarget objects to form an equal weighted portfolio based on the Insight objects.
SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
```

To view all the pre-built Portfolio Construction models, see Supported Models .

Model Structure

Portfolio Construction models should extend the `PortfolioConstructionModel` class or one of the supported models . Extensions of the `PortfolioConstructionModel` class should implement the `CreateTargets` method, which receives an array of `Insight` objects from the Alpha model at every time step and returns an array of `PortfolioTarget` objects. The Portfolio Construction model seeks to answer the question, "how many units should I buy based on the insight predictions I've been presented?".

If you don't override the `CreateTargets` method, the base class implementation calls the model's `IsRebalanceDue` , `DetermineTargetPercent` , and `GetTargetInsights` helper methods. The `GetTargetInsights` method, in turn, calls the model's `ShouldCreateTargetForInsight` method. You can override any of these helper methods. If you don't override the `CreateTargets` method from the `PortfolioConstructionModel` class, your class must at least override the `DetermineTargetPercent` method.

```C#
// Portfolio construction scaffolding class; basic method arguments.
class MyPortfolioConstructionModel : PortfolioConstructionModel
{
    // Create list of PortfolioTarget objects from Insights.
    public override List<PortfolioTarget> CreateTargets(QCAlgorithm algorithm, Insight[] insights)
    {
        return (List<PortfolioTarget>) base.CreateTargets(algorithm, insights);
    }

    // Determine if the portfolio should rebalance based on the provided rebalancing function.
    protected override bool IsRebalanceDue(Insight[] insights, DateTime algorithmUtc)
    {
        return base.IsRebalanceDue(insights, algorithmUtc);
    }

    // Determine the target percent for each insight.
    protected override Dictionary<Insight, double> DetermineTargetPercent(List<Insight> activeInsights)
    {
        return new Dictionary<Insight, double>();
    }

    // Get the target insights to calculate a portfolio target percent.
    // They will be piped to the DetermineTargetPercent method.
    protected override List<Insight> GetTargetInsights()
    {
        return base.GetTargetInsights();
    }

    // Determine if the portfolio construction model should create a target for this insight.
    protected override bool ShouldCreateTargetForInsight(Insight insight)
    {
        return base.ShouldCreateTargetForInsight(insight);
    }

    // Track universe changes.
    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        base.OnSecuritiesChanged(algorithm, changes);
    }
}
```

The Portfolio Construction model should remove expired insights from the Insight Manager . The `CreateTargets` definition of the base `PortfolioConstructionModel` class already removes them during each rebalance. Therefore, if you override the `CreateTargets` method and don't call the `CreateTargets` definition of the base class, your new method definition should remove expired insights from the Insight Manager.

The model should also remove all a security's insights from the Insight Manager when the security is removed from the universe . The `OnSecuritiesChanged` definition of the base `PortfolioConstructionModel` class already does this. Therefore, if you override the `OnSecuritiesChanged` method and don't call the `OnSecuritiesChanged` definition of the base class, your new method definition should remove the security's insights from the Insight Manager.

The `algorithm` argument that the methods receive is an instance of the base `QCAlgorithm` class, not your subclass of it.

You may use the `PortfolioBias` enumeration in the definition of Portfolio Construction model methods. The `PortfolioBias` enumeration has the following members:

To view a full example of a `PortfolioConstructionModel` subclass, see the EqualWeightingPortfolioConstructionModel in the LEAN GitHub repository.

Multi-Alpha Algorithms

If you add multiple Alpha models , each Alpha model receives the current slice in the order that you add the Alphas. The combined stream of Insight objects is passed to the Portfolio Construction model.

Each Portfolio Construction model has a unique method to combine Insight objects. The base `PortfolioConstructionModel` that

most PCM's inherit from doesn't combine information from Insight objects with the same `Symbol` - but just gets the most recent active insight. To combine the active insights differently, override the `GetTargetInsights` , and return all active insights. The `DetermineTargetPercent` method implements the combination criteria and determines the target for each `Symbol` .

```
C#

// Implement MultipleAlphaPortfolioConstructionModel to handle and utilize insights from multiple Alpha models.
// The get_target_insights method retrieves current active insights, and determine_target_percent allocates portfolio
weights accordingly for integrating and balancing multiple Alpha signals within the portfolio.
public class MultipleAlphaPortfolioConstructionModel : PortfolioConstructionModel
{
    protected override List<Insight> GetTargetInsights()
    {
        return Algorithm.Insights.GetActiveInsights(Algorithm.UtcTime).ToList();
    }

    protected override Dictionary<Insight, double> DetermineTargetPercent(List<Insight> activeInsights)
    {
        return new Dictionary<Insight, double>();
    }
}
```

Portfolio Targets

The Portfolio Construction model returns `PortfolioTarget` objects, which are passed to the Risk Management model.

To create a `PortfolioTarget` object based on a quantity, pass the `Symbol` and quantity to the `PortfolioTarget` constructor.

```
C#

// Create a new portfolio target for 1200 IBM shares.
var target = new PortfolioTarget("IBM", 1200);
```

To create a `PortfolioTarget` object based on a portfolio weight, call the `Percent` method. This method is only available for margin accounts.

```
C#

// Calculate target equivalent to 10% of portfolio value
var target = PortfolioTarget.Percent(algorithm, "IBM", 0.1);
```

To include more information in the `PortfolioTarget` object, pass a `tag` argument to the constructor or the `Percent` method.

The `CreateTargets` method of your Portfolio Construction model must return an array of `PortfolioTarget` objects.

```
C#

return new PortfolioTarget[] {  new PortfolioTarget("IBM", 1200)  };
```

Portfolio Target Collection

The `PortfolioTargetCollection` class is a helper class to manage `PortfolioTarget` objects. The class manages an internal dictionary that has the security `Symbol` as the key and a `PortfolioTarget` as the value.

Add Portfolio Targets

To add a `PortfolioTarget` to the `PortfolioTargetCollection` , call the `Add` method.

```
C#

_targetsCollection.Add(portfolioTarget);
```

To add a list of `PortfolioTarget` objects, call the `AddRange` method.

```C#
_targetsCollection.AddRange(portfolioTargets);
```

Check Membership

To check if a `PortfolioTarget` exists in the `PortfolioTargetCollection` , call the `Contains` method.

```C#
var targetInCollection = _targetsCollection.Contains(portfolioTarget);
```

To check if a Symbol exists in the `PortfolioTargetCollection` , call the `ContainsKey` method.

```C#
var symbolInCollection = _targetsCollection.ContainsKey(symbol);
```

To get all the Symbol objects, use the `Keys` property.

```C#
var symbols = _targetsCollection.Keys;
```

Access Portfolio Targets

To access the `PortfolioTarget` objects for a Symbol, index the `PortfolioTargetCollection` with the Symbol.

```C#
var portfolioTarget = _targetsCollection[symbol];
```

To iterate through the `PortfolioTargetCollection` , call the `GetEnumerator` method.

```C#
var enumerator = _targetsCollection.GetEnumerator();
```

To get all the `PortfolioTarget` objects, use the `Values` property

```C#
var portfolioTargets = _targetsCollection.Values;
```

Order Portfolio Targets by Margin Impact

To get an enumerable where position reducing orders are executed first and the remaining orders are executed in decreasing order value, call the `OrderByMarginImpact` method.

```C#
foreach (var target in _targetsCollection.OrderByMarginImpact(algorithm))
{
    // Place order
}
```

This method won't return targets for securities that have no data yet. This method also won't return targets for which the sum of

the current holdings and open orders quantity equals the target quantity.

Remove Portfolio Targets

To remove a `PortfolioTarget` from the `PortfolioTargetCollection` , call the `Remove` method.

```
C#
removeSuccessful = _targetsCollection.Remove(symbol);
```

To remove all the `PortfolioTarget` objects, call the `Clear` method.

```
C#
_targetsCollection.Clear();
```

To remove all the `PortfolioTarget` objects that have been fulfilled, call the `ClearFulfilled` method.

```
C#
_targetsCollection.ClearFulfilled(algorithm);
```

Rebalance Frequency

If you use a Portfolio Construction model that is a subclass of the `PortfolioConstructionModel` class, you can set the rebalancing frequency of the model with a function. The rebalancing function receives the Coordinated Universal Time (UTC) of the algorithm and should return the next rebalance UTC time or `null` . If the function returns `null` , the model doesn't rebalance unless the rebalance settings trigger a rebalance. For a full example of a custom rebalance function, see the PortfolioRebalanceOnCustomFuncRegressionAlgorithm .

If you use a Portfolio Construction model with the following characteristics, you can also set the rebalancing frequency of the model with a `TimeSpan` , `Resolution` , or DateRules :

- The model is a subclass of the `EqualWeightingPortfolioConstructionModel` class.
- The model constructor calls the `EqualWeightingPortfolioConstructionModel` constructor.
- The model doesn't override the `CreateTargets` method.

To check which of the pre-built Portfolio Construction models support this functionality, see Supported Models .

Rebalance Settings

By default, portfolio construction models create `PortfolioTarget` objects to rebalance the portfolio when any of the following events occur:

- The model's rebalance function signals it's time to rebalance
- The Alpha model emits new insights
- The universe changes

To disable rebalances when the Alpha model emits insights or when insights expire, set `RebalancePortfolioOnInsightChanges` to false.

```
C#
// Disable automatic portfolio rebalancing upon insight change, allowing for manual control over when portfolio
adjustments are made based on insights.
Settings.RebalancePortfolioOnInsightChanges = false;
```

To disable rebalances when security changes occur, set `RebalancePortfolioOnSecurityChanges` to false.

```
// Disable automatic portfolio rebalancing upon security change, allowing for manual control over when portfolio
adjustments are made based on security additions or removals.
Settings.RebalancePortfolioOnSecurityChanges = false;
```

Portfolio Optimizer Structure

Some portfolio construction models contain an optimizer that accepts the historical returns of each security and returns a list of optimized portfolio weights. Portfolio optimizer models must implement the `IPortfolioOptimizer` interface, which has an `Optimize` method.

```
// Implement an equal-weighted portfolio optimizer to assign equal weights to all securities, providing basic
diversification to reduce risk compared to a concentrated portfolio.
public class MyPortfolioOptimizer : IPortfolioOptimizer
{
    public double[] Optimize(double[,] historicalReturns, double[] expectedReturns = null, double[,] covariance =
null)
    {
        // Create weights
        //  For example, equal-weighting:
        int numAssets = historicalReturns.GetLength(1);
        var weights = Enumerable.Repeat(1.0 / numAssets, numAssets).ToArray();

        return weights;
    }
}
```

The following table describes the arguments the `Optimize` method accepts:

Argument	Data Type	Description	Default Value
`historicalReturns`	`double[,]`	Matrix of historical returns where each column represents a security and each row returns for the given date/time (size: K x N)	
`expectedReturns`	`double[]`	Array of double with the portfolio annualized expected returns (size: K x 1)	`null`
`covariance`	`double[,]`	Multi-dimensional array of double with the portfolio covariance of annualized returns (size: K x K)	`null`

The method should return a K x 1 array of double objects that represent the portfolio weights.

To view all the pre-built portfolio optimization algorithms, see Supported Optimizers .

To view a full example of an `IPortfolioOptimizer` implementation, see the MaximumSharpeRatioPortfolioOptimizer in the LEAN GitHub repository.

Track Security Changes

The Universe Selection model may select a dynamic universe of assets, so you should not assume a fixed set of assets in the

Portfolio Construction model. When the Universe Selection model adds and removes assets from the universe, it triggers an `OnSecuritiesChanged` event. In the `OnSecuritiesChanged` event handler, you can initialize the security-specific state or load any history required for your Portfolio Construction model. If you need to save data for individual securities, cast the `Security` object to a `dynamic` object and then save custom members to it .

```C#
class MyPortfolioConstructionModel : PortfolioConstructionModel{
    private List<Security> _securities = new List<Security>();

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        base.OnSecuritiesChanged(algorithm, changes);
        foreach (var security in changes.AddedSecurities)
        {
            // Store and manage Symbol-specific data
            var dynamicSecurity = security as dynamic;
            dynamicSecurity.Sma = SMA(security.Symbol, 20);

            _securities.Add(security);
        }

        foreach (var security in changes.RemovedSecurities)
        {
            if (_securities.Contains(security))
            {
                algorithm.DeregisterIndicator((security as dynamic).Sma);

                _securities.Remove(security);
            }
        }
    }
}
```

Universe Timing Considerations

If the Portfolio Construction model manages some indicators or consolidators for securities in the universe and the universe selection runs during the indicator sampling period or the consolidator aggregation period, the indicators and consolidators might be missing some data. For example, take the following scenario:

- The security resolution is minute
- You have a consolidator that aggregates the security data into daily bars to update the indicator
- The universe selection runs at noon

In this scenario, you create and warm-up the indicator at noon. Since it runs at noon, the history request that gathers daily data to warm up the indicator won't contain any data from the current day and the consolidator that updates the indicator also won't aggregate any data from before noon. This process doesn't cause issues if the indicator only uses the close price to calculate the indicator value (like the simple moving average indicator) because the first consolidated bar that updates the indicator will have the correct close price. However, if the indicator uses more than just the close price to calculate its value (like the True Range indicator), the open, high, and low values of the first consolidated bar may be incorrect, causing the initial indicator values to be incorrect.

Examples

The following examples demonstrate common practices for implementing the framework portfolio construction model.

Example 1: All-Weather Portfolio

The following algorithm uses `RiskParityPortfolioConstructionModel` to construct an all-weather portfolio that dissipates 1-year daily variance equally among ETFs that represent different asset classes, including stock market (SPY), bond (TLT), gold (GLD), oil (USO), and agricultural(DBA).

```C#
public class FrameworkPortfolioConstructionModelAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 1);

        // Add a universe of selected lists of ETFs that represent various asset classes.
        var etfs = new[] {
            QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA),     // stock market
            QuantConnect.Symbol.Create("TLT", SecurityType.Equity, Market.USA),     // bond
            QuantConnect.Symbol.Create("GLD", SecurityType.Equity, Market.USA),     // gold
            QuantConnect.Symbol.Create("USO", SecurityType.Equity, Market.USA),     // oil
            QuantConnect.Symbol.Create("DBA", SecurityType.Equity, Market.USA)      // agricultural
        };
        AddUniverseSelection(new ManualUniverseSelectionModel(etfs));
        // Emit insights for all selected ETFs.
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(1)));
        // Using risk parity PCM to construct an all-weather portfolio to dissipate risk.
        SetPortfolioConstruction(new RiskParityPortfolioConstructionModel());
    }
}
```

Example 2: Protective Position

The following algorithm implements an equal-weighting portfolio on the given insights. To hedge catastrophic downside risk, each position would order a protective call/put 10% OTM.

C#

```
public class FrameworkPortfolioConstructionModelAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);

        // Set to raw data normalization for the strike and underlying price fair comparison.
        UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw;
        // Add a universe of the market representative SPY.
        var spy = new[] {
            QuantConnect.Symbol.Create("SPY", SecurityType.Equity, Market.USA)
        };
        AddUniverseSelection(new ManualUniverseSelectionModel(spy));
        // Emit insights for all selected ETFs. Each insight lasts for 7 days for the option hedge expiry.
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(7)));
        // Using a custom PCM to control the order size and hedge option position order with a 10% OTM threshold.
        SetPortfolioConstruction(new ProtectivePositionPortfolioConstructionModel(0.1m));
    }

    private class ProtectivePositionPortfolioConstructionModel : PortfolioConstructionModel
    {
        private readonly decimal _threshold;

        public ProtectivePositionPortfolioConstructionModel(decimal threshold)
        {
            _threshold = threshold;
        }

        public override List<PortfolioTarget> CreateTargets(QCAlgorithm algorithm, Insight[] insights)
        {
            var targets = new List<PortfolioTarget>();
            if (insights.Length == 0)
            {
                return targets;
            }

            // Equally invest in each position group to dissipate the capital risk.
            var fundPerInsight = algorithm.Portfolio.TotalPortfolioValue / insights.Length;

            foreach (var insight in insights)
            {
                var underlying = insight.Symbol;
                // Hedge position option right: long position should purchase OTM put, while short should purchase
OTM call.
                var right = insight.Direction == InsightDirection.Up ? OptionRight.Put : OptionRight.Call;
                var threshold = insight.Direction == InsightDirection.Up ? 1m - _threshold : 1m + _threshold;

                // Select a protective option position that expires in 7 days.
                var hedge = algorithm.OptionChain(underlying)
                    .Where(x => x.Expiry < algorithm.Time.AddDays(7) && x.Right == right)
                    .OrderByDescending(x => x.Expiry)
                    .ThenBy(x => Math.Abs(x.Strike - x.UnderlyingLastPrice * threshold))
                    .First();
                // Request the protective position data for trading.
                var hedgeSymbol = algorithm.AddOptionContract(hedge).Symbol;

                // Each insight will be ordered by the position group of the underlying and the hedging option
positions.
                var contractMultiplier = algorithm.Securities[underlying].SymbolProperties.ContractMultiplier;
                var quantity = Math.Floor(fundPerInsight / contractMultiplier / (hedge.BidPrice +
hedge.UnderlyingLastPrice));
                targets.Add(new PortfolioTarget(underlying, (int)(quantity * contractMultiplier)));
                targets.Add(new PortfolioTarget(hedgeSymbol, (int)quantity));
            }

            return targets;
        }
    }
}
```

Portfolio Construction

Supported Models

Introduction

This page describes the pre-built Portfolio Construction models in LEAN. The number of models grows over time. To add a model to LEAN, make a pull request to the GitHub repository . If none of these models perform exactly how you want, create a custom Portfolio Construction model .

Null Model

The `NullPortfolioConstructionModel` is the default Portfolio Construction model. It doesn't return any `PortfolioTarget` objects. It's useful if you need to analyze an Alpha model in isolation.

```
C#
// Set portfolio construction to NullPortfolioConstructionModel to bypass custom logic, useful for simplifying
testing or initial development when you want to manually manage or skip automatic portfolio adjustments.
SetPortfolioConstruction(new NullPortfolioConstructionModel());
```

For more information about this model, see the class reference and implementation .

Equal Weighting Model

The `EqualWeightingPortfolioConstructionModel` assigns an equal share of the portfolio to the securities with active insights. This weighting scheme is useful for universe rotation based on simple portfolio strategies.

```
C#
// Set portfolio construction to EqualWeightingPortfolioConstructionModel to allocate capital equally across selected
securities, ensuring balanced exposure and reducing the influence of any single asset on the portfolio.
SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`resolution`	`Resolution`	Rebalancing frequency	`Resolution.Daily`

This model supports other data types for the rebalancing frequency argument. For more information about the supported types, see Rebalance Frequency .

This model removes expired insights from the Insight Manager during each rebalance. It also removes all insights for a security when the security is removed from the universe .

For more information about this model, see the class reference and implementation .

Confidence Weighted Model

The `ConfidenceWeightedPortfolioConstructionModel` generates target portfolio weights based on the `Insight. Confidence` for the last Insight of each Symbol. If the Insight has a direction of `InsightDirection. Up` , the model generates long targets. If the Insight has a direction of `InsightDirection. Down` , the model generates short targets. If the sum of all the last active

Insight per Symbol is greater than 1, the model factors down each target percent holdings proportionally so the sum is 1. The model ignores `Insight` objects that have no `Confidence` value.

```
// Use ConfidenceWeightedPortfolioConstructionModel to assign portfolio weights based on the confidence level of the
latest Insights, creating long or short positions accordingly, and normalizing weights if the total exceeds 1,
leveraging the strength of predictions for better allocation and risk management.
SetPortfolioConstruction(new ConfidenceWeightedPortfolioConstructionModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`resolution`	`Resolution`	Rebalancing frequency	`Resolution.Daily`
`portfolioBias`	`PortfolioBias`	The bias of the portfolio	`PortfolioBias. LongShor t`

This model supports other data types for the rebalancing frequency argument. For more information about the supported types, see Rebalance Frequency .

This model removes expired insights from the Insight Manager during each rebalance. It also removes all insights for a security when the security is removed from the universe .

For more information about this model, see the class reference and implementation .

Insight Weighting Model

The `InsightWeightingPortfolioConstructionModel` generates target portfolio weights based on the `Insight. Weight` for the last Insight of each Symbol. If the Insight has a direction of `InsightDirection. Up` , the model generates long targets. If the Insight has a direction of `InsightDirection. Down` , the model generates short targets. If the sum of all the last active Insight per Symbol is greater than 1, the model factors down each target percent holdings proportionally so the sum is 1. The model takes the absolute value of the `Weight` of each `Insight` object and ignores `Insight` objects that have no `Weight` value.

```
// Use InsightWeightingPortfolioConstructionModel to allocate capital based on the latest Insight weights, generating
long or short targets as per the Insight direction and adjusting allocations if total weights exceed 1, ensuring
portfolio weights reflect current market insights and maintains proper risk management.
SetPortfolioConstruction(new InsightWeightingPortfolioConstructionModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`resolution`	`Resolution`	Rebalancing frequency	`Resolution.Daily`
`portfolioBias`	`PortfolioBias`	The bias of the portfolio	`PortfolioBias. LongShor t`

This model supports other data types for the rebalancing frequency argument. For more information about the supported types, see Rebalance Frequency .

This model removes expired insights from the Insight Manager during each rebalance. It also removes all insights for a security when the security is removed from the universe .

For more information about this model, see the class reference and implementation .

Sector Weighting Model

The `SectorWeightingPortfolioConstructionModel` generates target portfolio weights based on the `CompanyReference.IndustryTemplateCode` provided by the US Fundamental dataset . The target percent holdings of each sector is 1/S where S is the number of sectors and the target percent holdings of each security is 1/N where N is the number of securities of each sector. If the insight has a direction of `InsightDirection. Up` , the model generates long targets. If the insight has a direction of `InsightDirection. Down` , the model generates short targets. The model ignores `Insight` objects for Symbols that have no `CompanyReference.IndustryTemplateCode` .

```
C#
// Use SectorWeightingPortfolioConstructionModel to ensure diversified sector exposure and balanced allocation,
aligning portfolio weights with sector insights while managing long and short positions based on insight direction.
SetPortfolioConstruction(new SectorWeightingPortfolioConstructionModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`resolution`	`Resolution`	Rebalancing frequency	`Resolution.Daily`

This model supports other data types for the rebalancing frequency argument. For more information about the supported types, see Rebalance Frequency .

This model removes expired insights from the Insight Manager during each rebalance. It also removes all insights for a security when the security is removed from the universe .

For more information about this model, see the class reference and implementation .

Accumulative Insight Model

The `AccumulativeInsightPortfolioConstructionModel` generates target portfolio weights based on all the active insights of a security. For each active Insight of direction `InsightDirection. Up` , it increases the position size by a fixed percent. For each active Insight of direction `InsightDirection. Down` , it decreases the position size by a fixed percent. For each active Insight of direction `InsightDirection.Flat` , it moves the position size towards 0 by a fixed percent.

```
C#
// Use AccumulativeInsightPortfolioConstructionModel to adjust portfolio weights based on the cumulative value of all
active insights, increasing position size for insights with UP direction to capitalize on expected upward movements,
while reducing position size with down or flat insights to mitigate potential losses.
SetPortfolioConstruction(new AccumulativeInsightPortfolioConstructionModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`rebalanceFunc`	`Func<DateTime, DateTime?>`	A function that receives the algorithm UTC time and returns the next expected rebalance time. If the function returns `null`, the portfolio doesn't rebalance.	`null`
`portfolioBias`	`PortfolioBias`	The bias of the portfolio	`PortfolioBias.LongShort`
`percent`	`double`	The percentage amount of the portfolio value to allocate to a single insight	0.03 (3%)

This model supports other data types for the rebalancing frequency argument. For more information about the supported types, see Rebalance Frequency .

This model removes expired insights from the Insight Manager during each rebalance. It also removes all insights for a security when the security is removed from the universe .

For more information about this model, see the class reference and implementation .

Mean Variance Optimization Model

The `MeanVarianceOptimizationPortfolioConstructionModel` seeks to build a portfolio with the least volatility possible and achieve a target return.

```
C#
// Use MeanVarianceOptimizationPortfolioConstructionModel to determine portfolio weights by optimizing for the best
risk-return trade-off balancing the expected returns against the risk of each security to construct a portfolio
aiming to maximize returns for a given level of risk or minimize risk for a given level of returns.
SetPortfolioConstruction(new MeanVarianceOptimizationPortfolioConstructionModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`rebalanceResolution`	`Resolution`	Rebalancing frequency	`Resolution.Daily`
`portfolioBias`	`PortfolioBias`	The bias of the portfolio	`PortfolioBias.LongShort`
`lookback`	`int`	Historical return lookback period	1
`period`	`int`	The time interval of history price to calculate the weight	63
`resolution`	`Resolution`	The resolution of the history price	`Resolution.Daily`
`targetReturn`	`double`	The target portfolio return	0.02 (2%)
`optimizer`	`IPortfolioOptimizer`	The portfolio optimization algorithm	`null`

This model supports other data types for the rebalancing frequency argument. For more information about the supported types, see Rebalance Frequency . If you don't provide an `optimizer` argument, the default one is the MinimumVariancePortfolioOptimizer with upper and lower weights that respect the `portfolioBias` .

This model removes expired insights from the Insight Manager during each rebalance. It also removes all insights for a security when the security is removed from the universe .

For more information about this model, see the class reference and implementation .

Black Litterman Optimization Model

The `BlackLittermanOptimizationPortfolioConstructionModel` receives `Insight` objects from multiple Alphas and combines them into a single portfolio. These multiple sources of Alpha models can be seen as the "investor views" required in the classical model.

```
C#
// Use BlackLittermanOptimizationPortfolioConstructionModel to incorporate market equilibrium portfolio and the
investor views to create an improved portfolio aligning with both market expectations and investor insights.
SetPortfolioConstruction(new BlackLittermanOptimizationPortfolioConstructionModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`rebalanceResolution`	`Resolution`	Rebalancing frequency	`Resolution.Daily`
`portfolioBias`	`PortfolioBias`	The bias of the portfolio	`PortfolioBias. LongShort`
`lookback`	`int`	Historical return lookback period	1
`period`	`int`	The time interval of history price to calculate the weight	63
`resolution`	`Resolution`	The resolution of the history price	`Resolution.Daily`
`riskFreeRate`	`double`	The risk free rate	0.0
`delta`	`double`	The risk aversion coefficient of the market portfolio	2.5
`tau`	`double`	The model parameter indicating the uncertainty of the CAPM prior	0.05
`optimizer`	`IPortfolioOptimizer`	The portfolio optimization algorithm	`null`

This model supports other data types for the rebalancing frequency argument. For more information about the supported types, see Rebalance Frequency . If you don't provide an `optimizer` argument, the default one is the MinimumVariancePortfolioOptimizer with upper and lower weights that respect the `portfolioBias` .

This model removes expired insights from the Insight Manager during each rebalance. It also removes all insights for a security

when the security is removed from the universe .

For more information about this model, see the class reference and implementation .

Mean Reversion Model

The `MeanReversionPortfolioConstructionModel` implements an on-line portfolio selection technique, named "On-Line Moving Average Reversion" (OLMAR). The basic idea is to represent multi-period mean reversion as "Moving Average Reversion" (MAR), which explicitly predicts next price relatives using moving averages and then forms portfolios with online learning techniques.

```
C#
// Use MeanReversionPortfolioConstructionModel to build a portfolio that capitalizes on the expected mean-reversion
of asset prices, adjusting allocations based on deviations from historical averages to potentially enhance returns by
exploiting price reversions.
SetPortfolioConstruction(new MeanReversionPortfolioConstructionModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`rebalanceResolution`	`Resolution`	Rebalancing frequency	`Resolution.Daily`
`portfolioBias`	`PortfolioBias`	The bias of the portfolio	`PortfolioBias. LongShor t`
`reversionThreshold`	`decimal`	Reversion threshold	1
`windowSize`	`int`	The window size of mean price	20
`resolution`	`Resolution`	The resolution of the history price	`Resolution.Daily`

This model supports other data types for the rebalancing frequency argument. For more information about the supported types, see Rebalance Frequency .

This model removes expired insights from the Insight Manager during each rebalance. It also removes all insights for a security when the security is removed from the universe .

For more information about this model, see the class reference and implementation .

Reference:

- Li, B., Hoi, S. C. (2012). On-line portfolio selection with moving average reversion. arXiv preprint arXiv:1206.4626 .

Risk Parity Model

The `RiskParityPortfolioConstructionModel` seeks to build a portfolio with the equal contribution of risk to the total portfolio risk from all assets.

```
C#
// Use RiskParityPortfolioConstructionModel to allocate portfolio weights based on risk contributions, ensuring that
each asset contributes equally to the portfolio's overall risk, aiming for balanced risk exposure and potentially
improved risk-adjusted returns.
SetPortfolioConstruction(new RiskParityPortfolioConstructionModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`rebalanceResolution`	`Resolution`	Rebalancing frequency	`Resolution.Daily`
`portfolioBias`	`PortfolioBias`	The bias of the portfolio	`PortfolioBias. LongShort`
`lookback`	`int`	Historical return lookback period	1
`period`	`int`	The time interval of history price to calculate the weight	252
`resolution`	`Resolution`	The resolution of the history price	`Resolution.Daily`
`optimizer`	`IPortfolioOptimizer`	The portfolio optimization algorithm	`null`

This model supports other data types for the rebalancing frequency argument. For more information about the supported types, see Rebalance Frequency . If you don't provide an `optimizer` argument, the default one is the RiskParityPortfolioOptimizer .

This model removes expired insights from the Insight Manager during each rebalance. It also removes all insights for a security when the security is removed from the universe .

For more information about this model, see the class reference and implementation .

Portfolio Construction

Supported Optimizers

Introduction

This page describes the pre-built Portfolio Optimizer models in LEAN. The number of models grows over time. To add a model to LEAN, make a pull request to the GitHub repository . If none of these models perform exactly how you want, create a custom Portfolio Optimizer model .

Maximum Sharpe Ratio Optimizer

The `MaximumSharpeRatioPortfolioOptimizer` seeks to maximize the portfolio Sharpe Ratio .

```
C#
var optimizer = new MaximumSharpeRatioPortfolioOptimizer();
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`lower`	`double`	The lower bounds on portfolio weights	-1
`upper`	`double`	The upper bounds on portfolio weights	1
`riskFreeRate`	`double`	The risk free rate	0

For more information about this model, see the class reference and implementation .

Minimum Variance Optimizer

The `MinimumVariancePortfolioOptimizer` seeks to minimize the portfolio variance and achieve a target return.

```
C#
var optimizer = new MinimumVariancePortfolioOptimizer();
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`lower`	`double`	The lower bounds on portfolio weights	-1
`upper`	`double`	The upper bounds on portfolio weights	1
`targetReturn`	`double`	The target portfolio return	0.02 (2%)

For more information about this model, see the class reference and implementation .

Unconstrained Mean Variance Optimizer

The `UnconstrainedMeanVariancePortfolioOptimizer` seeks to find the optimal risk-adjusted portfolio that lies on the efficient frontier.

```C#
var optimizer = new UnconstrainedMeanVariancePortfolioOptimizer();
```

For more information about this model, see the class reference and implementation .

Risk Parity Optimizer

The `RiskParityPortfolioOptimizer` seeks to equalize the individual risk contribution to the total portfolio risk from each asset.

```C#
var optimizer = new RiskParityPortfolioOptimizer();
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`lower`	`double`	The lower bounds on portfolio weights	1e-05
`upper`	`double`	The upper bounds on portfolio weights	`Double.MaxValue`

For more information about this model, see the class reference and implementation .

Algorithm Framework

Risk Management

Risk Management

Key Concepts

Introduction

The Risk Management model seeks to manage risk on the `PortfolioTarget` collection it receives from the Portfolio Construction model before the targets reach the Execution model. There are many creative ways to manage risk. Some examples of risk management include the following:

- *"Trailing Stop Risk Management Model"*
  Create and manage trailing stop-loss orders for open positions.
- *"Option Hedging Risk Management Model"*
  Purchase options to hedge large equity exposures.
- *"Sector Exposure Risk Management Model"*
  Reduce position sizes when overexposed to sectors or individual assets, keeping the portfolio within diversification requirements.
- *"Flash Crash Detection Risk Management Model"*
  Scan for strange market situations that might be precursors to a flash crash and attempt to protect the portfolio when they are detected.

Add Models

To set a Risk Management model, in the `Initialize` method, call the `AddRiskManagement` method.

```
C#
// Add the null risk management model, which doesn't affect the portfolio targets.
AddRiskManagement(new NullRiskManagementModel());
```

To view all the pre-built Risk Management models, see Supported Models .

Multi-Model Algorithms

To add multiple Risk Management models, in the `Initialize` method, call the `AddRiskManagement` method multiple times.

```
C#
// Add multiple Risk Management models to sequentially adjust portfolio targets, with each model refining the targets
passed from the previous model, managing risk across individual securities and sectors.
AddRiskManagement(new MaximumDrawdownPercentPerSecurity());
AddRiskManagement(new MaximumSectorExposureRiskManagementModel());
```

If you add multiple Risk Management models, the original collection of `PortfolioTarget` objects from the Portfolio Construction model is passed to the first Risk Management model. The risk-adjusted targets from the first Risk Management model are

passed to the second Risk Management model. The process continues sequentially until all of the Risk Management models have had an opportunity to adjust the targets.

Model Structure

Risk Management models should extend the `RiskManagementModel` class. Extensions of the `RiskManagementModel` class must implement the `ManageRisk` method, which receives an array of `PortfolioTarget` objects from the Portfolio Construction model at every time step and should return an array of risk-adjusted `PortfolioTarget` objects. The method should only return the adjusted targets, not all of targets. If the method creates a `PortfolioTarget` object to liquidate a security, cancel the security's insights to avoid re-entering the position.

```C#
// Extend the RiskManagementModel class by implementing the ManageRisk method, which receives PortfolioTarget objects
from the Portfolio Construction model and returns only the risk-adjusted targets.
class MyRiskManagementModel : RiskManagementModel
{
    // Adjust the portfolio targets and return them. If no changes emit nothing.
    public override List<PortfolioTarget> ManageRisk(QCAlgorithm algorithm, PortfolioTarget[] targets)
    {
        return new List<PortfolioTarget>();
    }

    // Optional: Be notified when securities change
    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        // Security additions and removals are pushed here.
        // This can be used for setting up algorithm state.
        // changes.AddedSecurities
        // changes.RemovedSecurities
    }
}
```

The `algorithm` argument that the methods receive is an instance of the base `QCAlgorithm` class, not your subclass of it.

To view a full example of a `RiskManagementModel` subclass, see the MaximumDrawdownPercentPerSecurity in the LEAN GitHub repository.

Track Security Changes

The Universe Selection model may select a dynamic universe of assets, so you should not assume a fixed set of assets in the Risk Management model. When the Universe Selection model adds and removes assets from the universe, it triggers an `OnSecuritiesChanged` event. In the `OnSecuritiesChanged` event handler, you can initialize the security-specific state or load any history required for your Risk Management model. If you need to save data for individual securities, cast the `Security` object to a `dynamic` object and then save custom members to it .

```
C#
class MyRiskManagementModel : RiskManagementModel{
    private List<Security> _securities = new List<Security>();

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        base.OnSecuritiesChanged(algorithm, changes);
        foreach (var security in changes.AddedSecurities)
        {
            // Store and manage Symbol-specific data
            var dynamicSecurity = security as dynamic;
            dynamicSecurity.Sma = SMA(security.Symbol, 20);

            _securities.Add(security);
        }

        foreach (var security in changes.RemovedSecurities)
        {
            if (_securities.Contains(security))
            {
                algorithm.DeregisterIndicator((security as dynamic).Sma);

                _securities.Remove(security);
            }
        }
    }
}
```

Portfolio Target Collection

The `PortfolioTargetCollection` class is a helper class to manage `PortfolioTarget` objects. The class manages an internal dictionary that has the security `Symbol` as the key and a `PortfolioTarget` as the value.

Add Portfolio Targets

To add a `PortfolioTarget` to the `PortfolioTargetCollection` , call the `Add` method.

```
C#
_targetsCollection.Add(portfolioTarget);
```

To add a list of `PortfolioTarget` objects, call the `AddRange` method.

```
C#
_targetsCollection.AddRange(portfolioTargets);
```

Check Membership

To check if a `PortfolioTarget` exists in the `PortfolioTargetCollection` , call the `Contains` method.

```
C#
var targetInCollection = _targetsCollection.Contains(portfolioTarget);
```

To check if a Symbol exists in the `PortfolioTargetCollection` , call the `ContainsKey` method.

```
C#
var symbolInCollection = _targetsCollection.ContainsKey(symbol);
```

To get all the Symbol objects, use the `Keys` property.

```
var symbols = _targetsCollection.Keys;
```

Access Portfolio Targets

To access the `PortfolioTarget` objects for a Symbol, index the `PortfolioTargetCollection` with the Symbol.

```
var portfolioTarget = _targetsCollection[symbol];
```

To iterate through the `PortfolioTargetCollection` , call the `GetEnumerator` method.

```
var enumerator = _targetsCollection.GetEnumerator();
```

To get all the `PortfolioTarget` objects, use the `Values` property

```
var portfolioTargets = _targetsCollection.Values;
```

Order Portfolio Targets by Margin Impact

To get an enumerable where position reducing orders are executed first and the remaining orders are executed in decreasing order value, call the `OrderByMarginImpact` method.

```
foreach (var target in _targetsCollection.OrderByMarginImpact(algorithm))
{
    // Place order
}
```

This method won't return targets for securities that have no data yet. This method also won't return targets for which the sum of the current holdings and open orders quantity equals the target quantity.

Remove Portfolio Targets

To remove a `PortfolioTarget` from the `PortfolioTargetCollection` , call the `Remove` method.

```
removeSuccessful = _targetsCollection.Remove(symbol);
```

To remove all the `PortfolioTarget` objects, call the `Clear` method.

```
_targetsCollection.Clear();
```

To remove all the `PortfolioTarget` objects that have been fulfilled, call the `ClearFulfilled` method.

```
_targetsCollection.ClearFulfilled(algorithm);
```

Universe Timing Considerations

If the Risk Management model manages some indicators or consolidators for securities in the universe and the universe selection runs during the indicator sampling period or the consolidator aggregation period, the indicators and consolidators might be missing some data. For example, take the following scenario:

- The security resolution is minute
- You have a consolidator that aggregates the security data into daily bars to update the indicator
- The universe selection runs at noon

In this scenario, you create and warm-up the indicator at noon. Since it runs at noon, the history request that gathers daily data to warm up the indicator won't contain any data from the current day and the consolidator that updates the indicator also won't aggregate any data from before noon. This process doesn't cause issues if the indicator only uses the close price to calculate the indicator value (like the simple moving average indicator) because the first consolidated bar that updates the indicator will have the correct close price. However, if the indicator uses more than just the close price to calculate its value (like the True Range indicator), the open, high, and low values of the first consolidated bar may be incorrect, causing the initial indicator values to be incorrect.

Examples

The following examples demonstrate some common practices for implementing a risk management model.

Example 1: Take Profit Stop Loss

The following algorithm trades 20-60 EMA crosses on the top 500 liquid US stocks in equal weighting. To minimize risk and capture early profit, we can implement both `MaximumUnrealizedProfitPercentPerSecurity` to take profit and `TrailingStopRiskManagementModel` to stop loss by trailing high price.

```
C#
public class FrameworkRiskManagementAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Add a universe of the most liquid stocks since their trend is more capital-supported.
        AddUniverseSelection(new QC500UniverseSelectionModel());
        // Emit insights on EMA cross, indicating the trend changes. We use short-term versus medium-term for more
trade opportunities.
        AddAlpha(new EmaCrossAlphaModel(20, 60, Resolution.Daily));
        // Equal weighting on each insight is needed to dissipate capital risk evenly.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());

        // Take profit at 10%.
        AddRiskManagement(new MaximumUnrealizedProfitPercentPerSecurity(0.1m));
        // Trailing stop loss at 5%.
        AddRiskManagement(new TrailingStopRiskManagementModel(0.05m));
    }
}
```

Example 2: Tail Value At Risk

The following algorithm implements a custom risk management model that liquidates the position if an asset has PnL lower than the tail value-at-risk (TVaR) throughout the insight to avoid large catastrophic losses. To calculate the TVaR, we create a log return indicator and use the following formula for the calculation: $\textrm{TVaR}_{\alpha} = \mu + \sigma \times \phi[\Phi^{-1}(\alpha)] / (1 - \alpha)$

```
C#
```

```
using Accord.Statistics;
using MathNet.Numerics.Distributions;

public class FrameworkRiskManagementAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 15);
        SetCash(1000000);

        // Add a universe of the most liquid stocks since their trend is more capital-supported.
        AddUniverseSelection(new QC500UniverseSelectionModel());
        // Emit insights all for selected stocks, rebalancing every 2 weeks.
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(14)));
        // Equal weighting on each insight is needed to dissipate capital risk evenly.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());

        // Liquidate on extreme catastrophic event.
        AddRiskManagement(new TailValueAtRiskRiskManagementModel(0.05d, 14d));
    }

    private class TailValueAtRiskRiskManagementModel : RiskManagementModel
    {
        // The alpha level on TVaR calculation.
        private readonly double _alpha;
        // The number of days that the insight signal lasts for.
        private readonly double _days;
        private Dictionary<Symbol, SymbolData> _logRetBySymbol = new();

        public TailValueAtRiskRiskManagementModel(double alpha = 0.05d, double numDays = 14d)
        {
            _alpha = alpha;
            _days = numDays;
        }

        // Adjust the portfolio targets and return them. If there are no changes, emit nothing.
        public override IEnumerable<PortfolioTarget> ManageRisk(QCAlgorithm algorithm, IPortfolioTarget[] _)
        {
            var targets = new List<PortfolioTarget>();

            foreach (var (symbol, security) in algorithm.Securities)
            {
                if (!security.Invested)
                {
                    continue;
                }

                var pnl = security.Holdings.UnrealizedProfitPercent;
                // If the %PnL is worse than the preset level TVaR, we liquidate it.
                if (pnl < GetTVaR(symbol))
                {
                    // Cancel insights to avoid reordering afterward.
                    algorithm.Insights.Cancel(new[] { symbol });
                    // Liquidate.
                    targets.Add(new PortfolioTarget(symbol, 0, tag: "Liquidate due to TVaR"));
                }
            }

            return targets;
        }

        private decimal GetTVaR(Symbol symbol)
        {
            if (!_logRetBySymbol.TryGetValue(symbol, out var symbolData))
            {
                return 0m;
            }

            // TVaR = \mu + \sigma * \phi[\Phi^{-1}(p)] / (1 - p)
            // Scale up to the days of the signal. By stochastic calculus, we multiply by sqrt(days).
            var dailyTVaR = symbolData.MeanLogRet + symbolData.SdLogRet * Math.Sqrt(_days) * Normal.PDF(0d, 1d,
Normal.InvCDF(0d, 1d, _alpha)) / (1d - _alpha);
            // We want the left side of the symmetric distribution.
            return -Convert.ToDecimal(dailyTVaR);
        }

        public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
        {
            foreach (var added in changes.AddedSecurities)
```

```
            {
                // Add SymbolData class to handle log returns.
                _logRetBySymbol[added.Symbol] = new SymbolData(algorithm, added.Symbol);
            }

            foreach (var removed in changes.RemovedSecurities)
            {
                // Stop subscription on the data to release computational resources.
                if (_logRetBySymbol.Remove(removed.Symbol, out var symbolData))
                {
                    symbolData.Dispose();
                }
            }
        }
    }

    private class SymbolData
    {
        private readonly QCAlgorithm _algorithm;
        private readonly Symbol _symbol;
        // Since the return is assumed log-normal, we use the log return indicator to calculate TVaR later.
        private LogReturn _logRet = new(1);
        // Set up a rolling window to save the log return for calculating the mean and SD for TVaR calculation.
        private RollingWindow<double> _window = new(252);

        public bool IsReady => _window.IsReady;

        public double MeanLogRet => _window.Average();

        public double SdLogRet => Measures.StandardDeviation(_window.ToArray());

        public SymbolData(QCAlgorithm algorithm, Symbol symbol)
        {
            _algorithm = algorithm;
            // Register the indicator for automatic updating for daily log returns.
            algorithm.RegisterIndicator(symbol, _logRet, Resolution.Daily);
            // Add a handler to save the log return to the rolling window.
            _logRet.Updated += (_algorithm, point) => _window.Add((double)point.Value);
            // Warm up the rolling window.
            var history = algorithm.History<TradeBar>(symbol, 253, Resolution.Daily);
            foreach (var bar in history)
            {
                _logRet.Update(bar.EndTime, bar.Close);
            }
        }

        public void Dispose()
        {
            // Stop subscription on the data to release computational resources.
            _algorithm.DeregisterIndicator(_logRet);
        }
    }
}
```

Risk Management

Supported Models

Introduction

This page describes the pre-built Risk Management models in LEAN. The number of models grows over time. To add a model to LEAN, make a pull request to the GitHub repository . If none of these models perform exactly how you want, create a custom Risk Management model .

Null Model

The `NullRiskManagementModel` is the default Risk Management model. It doesn't adjust any of the `PortfolioTarget` objects it receives from the Portfolio Construction model.

```
C#
// Adding the null risk management model doesn't affect the portfolio targets.
AddRiskManagement(new NullRiskManagementModel());
```

For more information about this model, see the class reference and implementation .

Maximum Security Drawdown Model

The `MaximumDrawdownPercentPerSecurity` model monitors the unrealized profit percentage of each security in the portfolio. When the percentage drops below a threshold relative to the opening price, it liquidates the position and cancels all insights in the Insight Manager that are for the security. This model can operate even when the Portfolio Construction model provides an empty list of `PortfolioTarget` objects.

```
C#
// Add risk management model to limit maximum drawdown per security, reducing position size to prevent further
losses.
AddRiskManagement(new MaximumDrawdownPercentPerSecurity());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`maximumDrawdownPercent`	`decimal`	The maximum percentage drawdown allowed for any single security holding	0.05 (5%)

For more information about this model, see the class reference and implementation .

Maximum Portfolio Drawdown Model

The `MaximumDrawdownPercentPortfolio` model monitors the portfolio drawdown . The drawdown can be relative to the starting portfolio value or the maximum portfolio value. When the portfolio value drops below a percentage threshold, the model liquidates the portfolio and cancels all insights in the Insight Manager . To liquidate the portfolio, the model must receive at least 1 `PortfolioTarget` after the drawdown threshold is passed. After the portfolio is liquidated, the model resets. This model can operate even when the Portfolio Construction model provides an empty list of `PortfolioTarget` objects.

```
// Add risk management model to limit maximum drawdown for the entire portfolio, protecting overall portfolio value
from significant losses.
AddRiskManagement(new MaximumDrawdownPercentPortfolio());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`maximumDrawdownPercent`	`decimal`	Maximum spread accepted comparing to current price in percentage	0.05 (5%)
`isTrailing`	`bool`	If `false` , the drawdown is relative to the starting value of the portfolio. If `true` , the drawdown is relative the last maximum portfolio value	`false`

For more information about this model, see the class reference and implementation .

Maximum Unrealized Profit Model

The `MaximumUnrealizedProfitPercentPerSecurity` model monitors the unrealized profit of each security in the portfolio. When the unrealized profit exceeds a profit threshold, it liquidates the position and cancels all insights in the Insight Manager that are for the security. This model can operate even when the Portfolio Construction model provides an empty list of `PortfolioTarget` objects.

```
// Add risk management model to secure gains by limiting the maximum unrealized profit for each security, locking in
profits before potential reversals.
AddRiskManagement(new MaximumUnrealizedProfitPercentPerSecurity());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`maximumUnrealizedProfitPercent`	`decimal`	The maximum percentage unrealized profit allowed for any single security holding	0.05 (5%)

For more information about this model, see the class reference and implementation .

Maximum Sector Exposure Model

The `MaximumSectorExposureRiskManagementModel` limits the absolute portfolio exposure in a each industry sector to a predefined maximum percentage. If the absolute portfolio exposure exceeds the maximum percentage, the weight of each Equity in the sector is scaled down so the sector doesn't exceed the maximum percentage. This process requires assets that are selected by Morningstar fundamental data . This model can operate even when the Portfolio Construction model provides an empty list of `PortfolioTarget` objects.

```
// Add risk management model to limit exposure to any single sector, ensuring diversification and reducing sector-specific risk.
AddRiskManagement(new MaximumSectorExposureRiskManagementModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`maximumSectorExposure`	`decimal`	The maximum exposure for any sector	0.2 (20%)

For more information about this model, see the class reference and implementation .

Trailing Stop Model

The `TrailingStopRiskManagementModel` monitors the drawdown of each security in the portfolio. When the peak-to-trough drawdown of the unrealized profit exceeds a threshold, it liquidates the position and cancels all insights in the Insight Manager that are for the security. This model can operate even when the Portfolio Construction model provides an empty list of `PortfolioTarget` objects.

```
// Add risk management model to implement trailing stops, which adjusts stop-loss levels as the price moves favorably, helping to lock in gains and limit losses.
AddRiskManagement(new TrailingStopRiskManagementModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`maximumDrawdownPercent`	`decimal`	The maximum percentage drawdown allowed for algorithm portfolio compared with the highest unrealized profit	0.05 (5%)

For more information about this model, see the class reference and implementation .

Algorithm Framework

Execution

Execution

Key Concepts

Introduction

The Execution model receives an array of risk-adjusted `PortfolioTarget` objects from the Risk Management model and places trades in the market to satisfy the targets. The Execution model only receives updates to the portfolio target share counts. It doesn't necessarily receive all of the targets at once.

Set Models

To set an Execution model, in the `Initialize` method, call the `SetExecution` method.

```
// Satisify portfolio targets by immediately sending market orders.
SetExecution(new ImmediateExecutionModel());
```

To view all the pre-built Execution models, see Supported Models .

Model Structure

Execution models should extend the `ExecutionModel` class. Extensions of the `ExecutionModel` must implement the `Execute` method, which receives an array of `PortfolioTarget` objects at every time step and is responsible for reaching the target portfolio as efficiently as possible. The Portfolio Construction model creates the `PortfolioTarget` objects, the Risk Management model may adjust them, and then the Execution model places the orders to fulfill them.

```C#
 // Basic Execution Model Scaffolding Structure Example
class MyExecutionModel : ExecutionModel {

   // Fill the supplied portfolio targets efficiently.
   public override void Execute(QCAlgorithm algorithm, IPortfolioTarget[] targets)
   {
      // NOP
   }

   //  Optional: Securities changes event for handling new securities.
   public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
   {
        // Security additions and removals are pushed here.
        // This can be used for setting up algorithm state.
        // changes.AddedSecurities
        // changes.RemovedSecurities
   }

   //  Optional: Order event for handling new order events.
   public override void OnOrderEvent(QCAlgorithm algorithm, OrderEvent orderEvent)
   {
        // Order events are pushed here.
        // This can be used for tracking order status or handling invalid and cancelled orders.
   }
}
```

The `algorithm` argument that the methods receive is an instance of the base `QCAlgorithm` class, not your subclass of it.

The following table describes the properties of the `PortfolioTarget` class that you may access in the Execution model:

Property	Data Type	Description
`Symbol`	`Symbol`	Asset to trade
`Quantity`	`decimal`	Number of units to hold

To view a full example of an `ExecutionModel` subclass, see the ImmediateExecutionModel in the LEAN GitHub repository.

Track Security Changes

The Universe Selection model may select a dynamic universe of assets, so you should not assume a fixed set of assets in the Execution model. When the Universe Selection model adds and removes assets from the universe, it triggers an `OnSecuritiesChanged` event. In the `OnSecuritiesChanged` event handler, you can initialize the security-specific state or load any history required for your Execution model. If you need to save data for individual securities, cast the `Security` object to a `dynamic` object and then save custom members to it .

```
C#
class MyExecutionModel : ExecutionModel{
    private List<Security> _securities = new List<Security>();

    public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
    {
        foreach (var security in changes.AddedSecurities)
        {
            // Store and manage Symbol-specific data
            var dynamicSecurity = security as dynamic;
            dynamicSecurity.Sma = SMA(security.Symbol, 20);

            _securities.Add(security);
        }

        foreach (var security in changes.RemovedSecurities)
        {
            if (_securities.Contains(security))
            {
                algorithm.DeregisterIndicator((security as dynamic).Sma);

                _securities.Remove(security);
            }
        }
    }
}
```

Portfolio Target Collection

The `PortfolioTargetCollection` class is a helper class to manage `PortfolioTarget` objects. The class manages an internal dictionary that has the security `Symbol` as the key and a `PortfolioTarget` as the value.

Add Portfolio Targets

To add a `PortfolioTarget` to the `PortfolioTargetCollection` , call the `Add` method.

```
C#
_targetsCollection.Add(portfolioTarget);
```

To add a list of `PortfolioTarget` objects, call the `AddRange` method.

```
C#
_targetsCollection.AddRange(portfolioTargets);
```

Check Membership

To check if a `PortfolioTarget` exists in the `PortfolioTargetCollection` , call the `Contains` method.

```
C#
var targetInCollection = _targetsCollection.Contains(portfolioTarget);
```

To check if a Symbol exists in the `PortfolioTargetCollection` , call the `ContainsKey` method.

```
C#
var symbolInCollection = _targetsCollection.ContainsKey(symbol);
```

To get all the Symbol objects, use the `Keys` property.

```
var symbols = _targetsCollection.Keys;
```

Access Portfolio Targets

To access the `PortfolioTarget` objects for a Symbol, index the `PortfolioTargetCollection` with the Symbol.

```
var portfolioTarget = _targetsCollection[symbol];
```

To iterate through the `PortfolioTargetCollection` , call the `GetEnumerator` method.

```
var enumerator = _targetsCollection.GetEnumerator();
```

To get all the `PortfolioTarget` objects, use the `Values` property

```
var portfolioTargets = _targetsCollection.Values;
```

Order Portfolio Targets by Margin Impact

To get an enumerable where position reducing orders are executed first and the remaining orders are executed in decreasing order value, call the `OrderByMarginImpact` method.

```
foreach (var target in _targetsCollection.OrderByMarginImpact(algorithm))
{
    // Place order
}
```

This method won't return targets for securities that have no data yet. This method also won't return targets for which the sum of the current holdings and open orders quantity equals the target quantity.

Remove Portfolio Targets

To remove a `PortfolioTarget` from the `PortfolioTargetCollection` , call the `Remove` method.

```
removeSuccessful = _targetsCollection.Remove(symbol);
```

To remove all the `PortfolioTarget` objects, call the `Clear` method.

```
_targetsCollection.Clear();
```

To remove all the `PortfolioTarget` objects that have been fulfilled, call the `ClearFulfilled` method.

```
_targetsCollection.ClearFulfilled(algorithm);
```

Universe Timing Considerations

If the Execution model manages some indicators or consolidators for securities in the universe and the universe selection runs during the indicator sampling period or the consolidator aggregation period, the indicators and consolidators might be missing some data. For example, take the following scenario:

- The security resolution is minute
- You have a consolidator that aggregates the security data into daily bars to update the indicator
- The universe selection runs at noon

In this scenario, you create and warm-up the indicator at noon. Since it runs at noon, the history request that gathers daily data to warm up the indicator won't contain any data from the current day and the consolidator that updates the indicator also won't aggregate any data from before noon. This process doesn't cause issues if the indicator only uses the close price to calculate the indicator value (like the simple moving average indicator) because the first consolidated bar that updates the indicator will have the correct close price. However, if the indicator uses more than just the close price to calculate its value (like the True Range indicator), the open, high, and low values of the first consolidated bar may be incorrect, causing the initial indicator values to be incorrect.

Examples

The following examples demonstrate some common practices for implementing the execution model.

Example 1: Iceberg Execution

The following algorithm simulates an account with large capital that equally holds the most liquid stocks. To hide the footprint and avoid a large market impact that might erode the profit margin, we can set up an iceberg execution system, submitting only 10% of the volume of the bid/ask side order book to the order direction.

```C#
public class FrameworkExecutionModelAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000000);

        // Add a universe of the most liquid stocks since their trend is more capital-supported.
        AddUniverseSelection(new QC500UniverseSelectionModel());
        // Emit insights all for selected stocks.
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(7)));
        // Equal weighting on each insight is needed to dissipate capital risk evenly.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());

        // Iceberg ordering to hide traces and avoid market impact.
        // Since quote data will be used, make sure the asset class and resolution are compatible.
        SetExecution(new IcebergExecutionModel(0.1m));
    }

    private class IcebergExecutionModel : ExecutionModel
    {
        private readonly PortfolioTargetCollection _targetsCollection = new PortfolioTargetCollection();
        // The maximum order size taken from the order book in percentage.
        private readonly decimal _maximumOrderQuantityPercentVolume;

        public IcebergExecutionModel(decimal orderPercentVolume = 0.1m)
        {
            _maximumOrderQuantityPercentVolume = orderPercentVolume;
        }

        public override void Execute(QCAlgorithm algorithm, IPortfolioTarget[] targets)
        {
            //Update the complete set of portfolio targets with the new targets
            _targetsCollection.AddRange(targets);

            if (!_targetsCollection.IsEmpty)
            {
                foreach (var target in _targetsCollection.OrderByMarginImpact(algorithm))
                {
                    var symbol = target.Symbol;

                    // Calculate the remaining quantity to be ordered
                    var unorderedQuantity = OrderSizing.GetUnorderedQuantity(algorithm, target);
                    // Adjust order size to respect maximum order size based on a percentage of the current volume
                    var orderSize = GetOrderSizeForPercentVolume(algorithm.Securities[symbol],
_maximumOrderQuantityPercentVolume, unorderedQuantity);

                    if (orderSize != 0)
                    {
                        algorithm.MarketOrder(symbol, orderSize);
                    }
                }

                _targetsCollection.ClearFulfilled(algorithm);
            }
        }

        private static decimal GetOrderSizeForPercentVolume(Security security, decimal maximumPercentCurrentVolume,
decimal desiredOrderSize)
        {
            // Take N% from the order book according to the order direction.
            var maxOrderSize = maximumPercentCurrentVolume * (desiredOrderSize > 0 ? security.BidSize :
security.AskSize);
            var orderSize = Math.Min(maxOrderSize, Math.Abs(desiredOrderSize));

            return Math.Sign(desiredOrderSize) * OrderSizing.AdjustByLotSize(security, orderSize);
        }
    }
}
```

Example 2: More Favorable Than Signal Time

Some algorithms are signaling based on daily close data. However, their orders often emit when the next market opens, making the filling price slip. To avoid this slippage negatively impacting your expectations, we can construct a custom execution model

to place orders only when the price is more favorable than the last close price. Although we did not set a time limit in this example, you may consider doing so.

C#

```
public class FrameworkExecutionModelAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 1);

        // Add a universe of the most liquid stocks since their trend is more capital-supported.
        AddUniverseSelection(new QC500UniverseSelectionModel());
        // Emit insights all for selected stocks.
        AddAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(7)));
        // Equal weighting on each insight is needed to dissipate capital risk evenly.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());

        // Only place an order if the price is more favorable than the price at the insight signal.
        SetExecution(new FavorableExecutionModel());
    }

    private class FavorableExecutionModel : ExecutionModel
    {
        private readonly PortfolioTargetCollection _targetsCollection = new PortfolioTargetCollection();
        private Dictionary<Symbol, decimal> _lagPriceBySymbol = new();

        public override void Execute(QCAlgorithm algorithm, IPortfolioTarget[] targets)
        {
            _targetsCollection.AddRange(targets);
            if (!_targetsCollection.IsEmpty)
            {
                foreach (var target in _targetsCollection.OrderByMarginImpact(algorithm))
                {
                    var symbol = target.Symbol;
                    var security = algorithm.Securities[symbol];

                    // Calculate the remaining quantity to be ordered
                    var quantity = OrderSizing.GetUnorderedQuantity(algorithm, target, security, true);

                    if (quantity != 0)
                    {
                        if
(BuyingPowerModelExtensions.AboveMinimumOrderMarginPortfolioPercentage(security.BuyingPowerModel, security, quantity,
                            algorithm.Portfolio, algorithm.Settings.MinimumOrderMarginPortfolioPercentage))
                        {
                            // Cache the price at signal emission to compare if the current price is more favorable.
                            if (!_lagPriceBySymbol.TryGetValue(symbol, out var lastPrice))
                            {
                                var history = algorithm.History<TradeBar>(symbol, 1, Resolution.Daily);
                                _lagPriceBySymbol[symbol] = lastPrice = history.Last().Close;
                            }

                            // Only order if the price is more favorable than the price at signal emission.
                            if ((security.Price <= lastPrice && quantity > 0) || (security.Price >= lastPrice &&
quantity < 0))
                            {
                                algorithm.MarketOrder(security, quantity);
                            }
                        }
                        else if (!PortfolioTarget.MinimumOrderMarginPercentageWarningSent.HasValue)
                        {
                            // will trigger the warning if it has not already been sent
                            PortfolioTarget.MinimumOrderMarginPercentageWarningSent = false;
                        }
                    }
                }

                _targetsCollection.ClearFulfilled(algorithm);
            }
        }
    }
}
```

Execution

Supported Models

Introduction

This page describes the pre-built Execution models in LEAN. The number of models grows over time. To add a model to LEAN, make a pull request to the GitHub repository . If none of these models perform exactly how you want, create a custom Execution model .

Null Model

The `NullExecutionModel` doesn't place any trades.

```
C#
// Set the execution model to NullExecutionModel, which disables execution functionality for debugging or when the
execution logic is not needed for the current strategy.
SetExecution(new NullExecutionModel());
```

For more information about this model, see the class reference and implementation .

Immediate Model

The `ImmediateExecutionModel` is the default Execution model. It uses market orders to immediately fill portfolio targets. It's similar to placing market orders in line with your algorithm logic.

```
C#
// Use ImmediateExecutionModel for executing trades instantly at market price to minimize execution delay. This may
incur extra slippage.
SetExecution(new ImmediateExecutionModel());
```

For more information about this model, see the class reference and implementation .

Spread Model

The `SpreadExecutionModel` executes trades with market orders if the exchange is open and the bid-ask spread is within a threshold percentage. The model only works if the security subscription provides QuoteBar data. The spread percentage is calculated as

$$\frac{a - b}{p}$$

where a is the ask price, b is the bid price, and p is the price.

```
C#
// Use SpreadExecutionModel to manage trade execution based on bid-ask spreads, aiming to optimize trade execution
costs and reduce slippage.
SetExecution(new SpreadExecutionModel());
```

The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`acceptingSpreadPercent`	`decimal`	Maximum spread accepted comparing to current price in percentage	0.005 (0.5%)

For more information about this model, see the class reference and implementation .

Standard Deviation Model

The `StandardDeviationExecutionModel` seeks to fill orders when the price is more than 2 standard deviations lower than the average price over a trailing period. The intent is to find dips in the market to place trades. In strongly trending markets, this procedure can result in delayed trade placement since it might be a while before the next price spike or dip.

```
// Use StandardDeviationExecutionModel to execute trades based on volatility measures, aiming to adapt trade sizes
and timing according to market volatility.
SetExecution(new StandardDeviationExecutionModel());
```

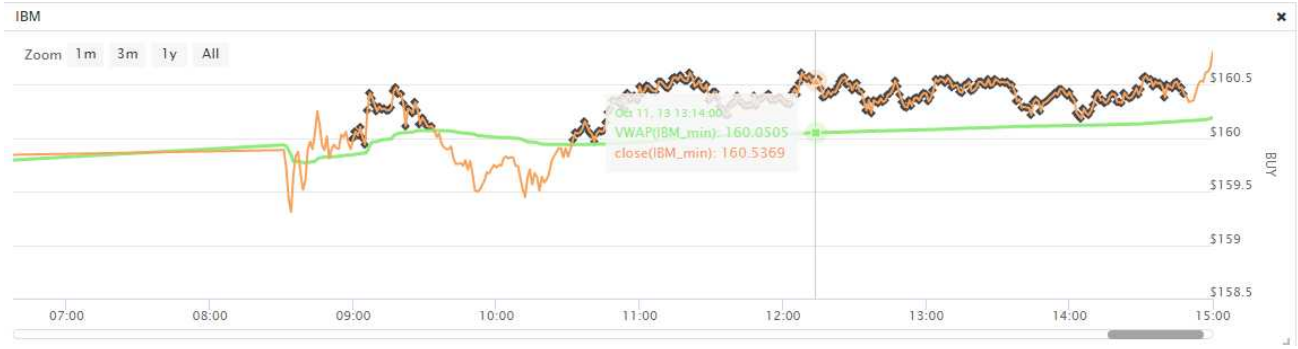
The following table describes the arguments the model accepts:

Argument	Data Type	Description	Default Value
`period`	`decimal`	Period of the standard deviation indicator	60
`deviations`	`int`	The number of deviations away from the mean before submitting an order	2
`resolution`	`Resolution`	The resolution of the STD and SMA indicators	`Resolution.Minute`

For more information about this model, see the class reference and implementation .

VWAP Model

The `VolumeWeightedAveragePriceExecutionModel` works to fill your orders at or better than the volume-weighted average price (VWAP) for the trading day. This is a best-effort algorithm, so no guarantee can be made that it will reach the VWAP.



VWAP Execution Model Fill Placements

The model uses market orders and it only works if the security subscription provides intraday data. It sets a maximum order size of 1% of the current bar's volume, but you can update the maximum order size through the `MaximumOrderQuantityPercentVolume` member.

```C#
// Use VolumeWeightedAveragePriceExecutionModel to execute trades based on volume-weighted average price, aiming to
minimize market impact and achieve better execution prices.
SetExecution(new VolumeWeightedAveragePriceExecutionModel());
```

For more information about this model, see the class reference and implementation .

Algorithm Framework

Hybrid Algorithms

Introduction

Classic style algorithms can also use Algorithm Framework modules. This allows you to get the best of both styles of algorithm design, combining easily pluggable modules with the superior control of classic format.

Examples of popular use-cases of a hybrid approach are:

- Using framework universe selection models as the assets to select.
- Using portfolio construction models to decide portfolio allocations.
- Using a risk control model for free portfolio risk monitoring.
- Passing data between modules freely without concerns of the module interface.

Universe Selection

You can add one or more Framework Universe Selection Models to your algorithm and it will operate normally. You can also combine it with the `AddUniverse` method of the classic approach. The following example initializes a classic algorithm with framework universe selection models:

```
C#
// Use a hybrid approach by combining Framework Universe Selection Models with classic methods to leverage both the
modular flexibility of Framework modules and the control of classic design. This allows for better universe
selection, portfolio allocation, and risk management.
public override void Initialize()
{
    UniverseSettings.Asynchronous = true;
    SetUniverseSelection(new ManualUniverseSelectionModel(QuantConnect.Symbol.Create("SPY", SecurityType.Equity,
Market.USA)));
    AddUniverseSelection(new ManualUniverseSelectionModel(QuantConnect.Symbol.Create("AAPL", SecurityType.Equity,
Market.USA)));
    AddUniverse(Universe.Top(5));
}
```

Alpha

You can add one or multiple Alpha models to your classic algorithm and place the orders using `Insight` objects without a Portfolio Construction model. To receive the collection of `Insight` objects in a classic algorithm, implement the `InsightsGenerated` event handler:

```
C#
// Add one or multiple Alpha models to a classic algorithm allowing for direct trading based on generated Insight
objects, bypassing the need for a Portfolio Construction model.
public override void Initialize()
{
    AddAlpha(new EmaCrossAlphaModel());
    // The InsightsGenerated event handler is used to receive and process the collection of insights, facilitating
flexible and dynamic trading strategies.
    InsightsGenerated += OnInsightsGenerated;
}
private void OnInsightsGenerated(IAlgorithm algorithm, GeneratedInsightsCollection insightsCollection)
{
    var insights = insightsCollection.Insights;
}
```

Portfolio Construction

You can add a Portfolio Construction model to your classic algorithm and have it place orders without returning `Insight` objects from an Alpha model. To emit insights without an Alpha model, in the `OnData` method, call the `EmitInsights` method.

The following example uses a Portfolio Construction Framework model with `EmitInsights` method in a classic algorithm:

```C#
// Add a Portfolio Construction model to place orders directly without relying on Alpha-generated insights.
public override void Initialize()
{
    SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
}
// Use the EmitInsights method in the OnData method to provide trading signals for executing trades, ensuring
balanced investment with the chosen Portfolio Construction model.
public override void OnData(Slice slice)
{
 EmitInsights(new Insight("GOOG", TimeSpan.FromMinutes(20), InsightType.Price, InsightDirection.Up));
 EmitInsights(new []{
  new Insight("AAPL", TimeSpan.FromMinutes(20), InsightType.Price, InsightDirection.Up),
  new Insight("MSFT", TimeSpan.FromMinutes(20), InsightType.Price, InsightDirection.Up)
 });
}
```

Risk Management

Some Risk Management Models don't require a Portfolio Construction model to provide `PortfolioTarget` objects, allowing them to directly monitor the portfolio holdings and liquidate positions when neccessary. To see which pre-built Risk Management models don't need the Portfolio Construction model to provide `PortfolioTarget` objects, see Supported Models .

You can add one or more Risk Management Models to your algorithm and it will operate normally. The following example initializes a classic algorithm with framework risk management models:

```C#
// Add Risk Management models to monitor portfolio risk directly, without needing PortfolioTargets from a Portfolio
Construction model. These models can independently manage holdings and trigger liquidations as needed.
public override void Initialize()
{
    AddRiskManagement(new MaximumDrawdownPercentPerSecurity(0.05m));
    AddRiskManagement(new MaximumUnrealizedProfitPercentPerSecurity(0.1m));
}
```

Execution

Execution models can place orders for your strategy instead of placing them manually. LEAN routes PortfolioTarget objects to the `Execute` method of the Execution Model to place orders.

The following example uses a Framework Execution Model in a classic style algorithm:

```C#
// Use SetExecution to place orders via the Execution Model, which routes PortfolioTarget objects to its
// execute method, instead of placing orders manually. This ensures automated and consistent order execution.
public override void Initialize()
{
    // Execute the market order imediately to fill the portfolio targets.
    SetExecution(new ImmediateExecutionModel());
}
```

Universe Timing Considerations

If the Alpha, Portfolio Construction, Risk Management, or Execution model manages some indicators or consolidators for securities in the universe and the universe selection runs during the indicator sampling period or the consolidator aggregation

period, the indicators and consolidators might be missing some data. For example, take the following scenario:

- The security resolution is minute
- You have a consolidator that aggregates the security data into daily bars to update the indicator
- The universe selection runs at noon

In this scenario, you create and warm-up the indicator at noon. Since it runs at noon, the history request that gathers daily data to warm up the indicator won't contain any data from the current day and the consolidator that updates the indicator also won't aggregate any data from before noon. This process doesn't cause issues if the indicator only uses the close price to calculate the indicator value (like the simple moving average indicator) because the first consolidated bar that updates the indicator will have the correct close price. However, if the indicator uses more than just the close price to calculate its value (like the True Range indicator), the open, high, and low values of the first consolidated bar may be incorrect, causing the initial indicator values to be incorrect.

Examples

The following examples demonstrate common practices for implementing hybrid algorithms.

Example 1: Corresponding Alpha

The following algorithm trades two separate logics:

- EMA cross on the 20 most liquid stocks
- Buy and hold the top 20 weighted SPY constituents

To do so, we can save the universe and allow an access point in the corresponding alpha model to filter the signals that are only applicable to the selected equities from each universe.

C#

```
public class FrameworkHybridAlgorithm : QCAlgorithm
{
    public Universe Universe1, Universe2;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Add a universe of the most liquid stocks since their trend is more capital-supported to trend EMA cross.
        Universe1 = AddUniverse(LiquidSelection);
        // Emit insights if the trend of selected liquid stocks changes.
        AddAlpha(new CustomEmaCrossAlphaModel(this));

        // Add another universe set to pick the most weighted SPY constituents since they usually have excess return
compared to the rest.
        Universe2 = AddUniverse(Universe.ETF("SPY", Market.USA, UniverseSettings, EtfConstituentFilter));
        // Emit insights for all selected SPY constituents.
        AddAlpha(new CustomConstantAlphaModel(this));

        // Equal weighting on each insight is needed to dissipate capital risk evenly.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());
    }

    private IEnumerable<Symbol> LiquidSelection(IEnumerable<Fundamental> fundamentals)
    {
        // Select the top 20 liquid equities for EMA cross trading.
        return (from f in fundamentals
            orderby f.DollarVolume descending
            select f.Symbol).Take(20);
    }

    private IEnumerable<Symbol> EtfConstituentFilter(IEnumerable<ETFConstituentUniverse> constituents)
    {
        // Select the top 20 weighted stocks of SPY constituents to hold.
        return (from c in constituents
```

```
            where c.Weight.HasValue
            orderby c.Weight.Value descending
            select c.Symbol).Take(20);
    }

    private class CustomEmaCrossAlphaModel : EmaCrossAlphaModel
    {
        private FrameworkHybridAlgorithm _algorithm;
        private readonly int _fastPeriod, _slowPeriod;
        private readonly Resolution _resolution;

        public CustomEmaCrossAlphaModel(FrameworkHybridAlgorithm algorithm, int fastPeriod = 20, int slowPeriod = 60,
Resolution resolution = Resolution.Daily)
            : base(fastPeriod, slowPeriod, resolution)
        {
            _algorithm = algorithm;
            _fastPeriod = fastPeriod;
            _slowPeriod = slowPeriod;
            _resolution = resolution;
        }

        public override void OnSecuritiesChanged(QCAlgorithm algorithm, SecurityChanges changes)
        {
            foreach (var added in changes.AddedSecurities)
            {
                // Only trade the liquid universe for this alpha model.
                if (_algorithm.Universe1.Selected.Contains(added.Symbol))
                {
                    SymbolData symbolData;
                    if (!SymbolDataBySymbol.TryGetValue(added.Symbol, out symbolData))
                    {
                        SymbolDataBySymbol[added.Symbol] = new SymbolData(added, _fastPeriod, _slowPeriod, algorithm,
_resolution);
                    }
                    else
                    {
                        // A security that was already initialized was re-added, reset the indicators.
                        symbolData.Fast.Reset();
                        symbolData.Slow.Reset();
                    }
                }
            }

            foreach (var removed in changes.RemovedSecurities)
            {
                SymbolData symbolData;
                if (SymbolDataBySymbol.TryGetValue(removed.Symbol, out symbolData))
                {
                    // clean up our consolidators.
                    symbolData.RemoveConsolidators();
                    SymbolDataBySymbol.Remove(removed.Symbol);
                }
            }
        }
    }

    private class CustomConstantAlphaModel : ConstantAlphaModel
    {
        private FrameworkHybridAlgorithm _algorithm;

        public CustomConstantAlphaModel(FrameworkHybridAlgorithm algorithm)
            // Daily insight length due to daily signals.
            : base(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(1))
        {
            _algorithm = algorithm;
        }

        public override IEnumerable<Insight> Update(QCAlgorithm algorithm, Slice data)
        {
            var insights = base.Update(algorithm, data);
            // Only trade the selected SPY constituents universe for this alpha model.
            return insights.Where(x => _algorithm.Universe2.Selected.Contains(x.Symbol));
        }
    }
}
```

Example 2: Bracket Order On Insight

The following algorithm uses framework universe selection and alpha model to filter the universe and generate insights. However, the portfolio construction model cannot handle bracket orders easily. Hence, we will access the active insights to place bracket orders using the classic implementation, with stop losses at 1% and take profits at 2%.

C#

```
public class FrameworkHybridAlgorithm : QCAlgorithm
{
    private List<Insight> _orderedInsights = new();

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(1000000);

        // Add a universe of the most liquid stocks since their trend is more capital-supported to trend EMA cross.
        AddUniverse(LiquidSelection);
        // Emit insights if the trend of selected liquid stocks changes.
        AddAlpha(new EmaCrossAlphaModel(20, 60, Resolution.Daily));
    }

    private IEnumerable<Symbol> LiquidSelection(IEnumerable<Fundamental> fundamentals)
    {
        // Select the top 20 liquid equities for EMA cross trading.
        return (from f in fundamentals
            orderby f.DollarVolume descending
            select f.Symbol).Take(20);
    }

    public override void OnData(Slice slice)
    {
        // Exit positions if insight is expired.
        foreach (var insight in new List<Insight>(_orderedInsights))
        {
            // If the insight is expired or profit taken or stop loss
            if (insight.IsExpired(UtcTime) || !Portfolio[insight.Symbol].Invested)
            {
                // Discontinue the insight.
                Insights.Cancel(new[] { insight });
                // Liquidate any positions and any open (bracket) orders.
                Liquidate(insight.Symbol);
                // Remove from the cache.
                _orderedInsights.Remove(insight);
            }
        }

        // Place orders for active insights if not yet ordered.
        var activeInsights = Insights.GetActiveInsights(UtcTime);
        foreach (var insight in activeInsights)
        {
            if (!_orderedInsights.Contains(insight))
            {
                // Position sizing by given weight or equal weighting to dissipate capital risk.
                var size = 0d;
                if (insight.Weight.HasValue)
                {
                    size = insight.Weight.Value;
                }
                else
                {
                    size = 1d / activeInsights.Count;
                }
                SetHoldings(insight.Symbol, size);
                // Add to cache to avoid re-order.
                _orderedInsights.Add(insight);
            }
        }
    }

    public override void OnOrderEvent(OrderEvent orderEvent)
    {
        if (orderEvent.Status == OrderStatus.Filled)
        {
            if (orderEvent.Ticket.OrderType == OrderType.Market && Portfolio[orderEvent.Symbol].Quantity != 0)
            {
                // Stop loss order at 1%.
                var stopPrice = orderEvent.FillQuantity > 0m ? orderEvent.FillPrice * 0.99m : orderEvent.FillPrice *
1.01m;
```

```
                StopMarketOrder(orderEvent.Symbol, -Portfolio[orderEvent.Symbol].Quantity, stopPrice);
                // Take profit order at 2%.
                var takeProfitPrice = orderEvent.FillQuantity > 0m ? orderEvent.FillPrice * 1.02m :
orderEvent.FillPrice * 0.98m;
                LimitOrder(orderEvent.Symbol, -Portfolio[orderEvent.Symbol].Quantity, takeProfitPrice);
            }
            else if (orderEvent.Ticket.OrderType == OrderType.StopMarket || orderEvent.Ticket.OrderType ==
OrderType.Limit)
            {
                // Cancel any open order if stop loss or take profit order filled.
                Transactions.CancelOpenOrders();
            }
        }
    }
}
```

Algorithm Framework

Insight Manager

Introduction

The `InsightManager` tracks all of the Insight objects in your algorithm. It's an `InsightCollection`, which stores all of your insights in an internal dictionary that has the security `Symbol` as the key and a list of `Insight` objects as the value. You can access the manager anywhere in your algorithm where you have reference to the algorithm class. If you want to use an `InsightCollection` in your algorithm that's seperate from the `InsightManager`, create a new `InsightCollection`.

Check Membership

To check if an insight exists in the `InsightManager`, call the `Contains` method.

```
C#

// Check the algorithms central insight collection (InsightManager) to avoid emitting duplicates, reducing the
processing work.
if (algorithm.Insights.Contains(insight))
{

}
```

To check if the `InsightManager` has an insight for a `Symbol`, call the `ContainsKey` method.

```
C#

// Check the algorithms central insight collection (InsightManager) if there's already been an insight for this
symbol across the other alphas.
if (algorithm.Insights.ContainsKey(symbol))
{

}
```

To check if the `InsightManager` has active insights for a `Symbol` at a specific Coordinated Universal Time (UTC), call the `HasActiveInsights` method.

```
C#

// Check if there are active insights for a specific symbol at the given time to determine if alpha should emit more
insights.
if (algorithm.Insights.HasActiveInsights(symbol, utcTime))
{

}
```

Get Insights

To get the insights for a `Symbol`, index the `InsightManager` with the `Symbol`.

```
C#

if (algorithm.Insights.ContainsKey(symbol))
{
    var insights = algorithm.Insights[symbol];
}
```

To get the insights that pass a filter, call the `GetInsights` method.

```
var insights = algorithm.Insights.GetInsights(insight => insight.Direction == InsightDirection.Up);
```

To iterate through the `InsightManager` , call the `GetEnumerator` method.

```
var enumerator = algorithm.insights.get_enumerator();
```

To get all of the insights that will be active at a certain UTC time, call the `GetActiveInsights` method.

```
var activeInsights = algorithm.Insights.GetActiveInsights(utcTime);
```

Get the Next Expiry Time

To get the next insight expiry time in UTC, call the `GetNextExpiryTime` method.

```
var nextExpiry = algorithm.Insights.GetNextExpiryTime();
```

Remove Insights

Only the Portfolio Construction model should remove insights from the `InsightManager` . It should remove insights when the insights expire and when the corresponding security leaves the universe .

To remove an insight from the `InsightManager` , call the `Remove` method.

```
var removeSuccessful = algorithm.Insights.Remove(insight);
```

To remove all the insights for a set of `Symbol` objects, pass a list of `Symbol` objects the `Clear` method.

```
algorithm.insights.clear(symbols);
```

To remove all the insights that will be expired at a certain UTC time, call the `RemoveExpiredInsights` method.

```
var expiredInsights = algorithm.Insights.RemoveExpiredInsights(utcTime);
```

Cancel Insights

In some cases, you may want to cancel an Insight. For example, if a Risk Management model in your algorithm liquidates a security, it should also cancel all of the active insights for the security. If you don't cancel the insights, the Portfolio Construction model might create a new PortfolioTarget to re-enter the position.

Another example of a situtation where you would want to cancel an Insight is when an Alpha model in your algorithm determines the trading opportunity has pre-maturely ended. For instance, say you want your algorithm to enter a long position in a security when its Relative Strength Index (RSI) moves below 20 and then liquidate the position when the security's RSI moves above 30. If you emit an Insight that has a duration of 30 days when the RSI moves below 20 but the RSI moves above 30 in 10 days, you

should cancel the Insight when the RSI moves above 30.

To cancel insights, call the `Cancel` / `Expire` method with a list of `Insight` objects.

```
algorithm.insights.cancel(insights);
```

To cancel all the insights for some securities, call the `Cancel` / `Expire` method with a list of `Symbol` objects.

```
algorithm.insights.cancel(symbols);
```

When you cancel an active insight, it's `CloseTimeUtc` property is set to one second into the past.

Add Insights to Self-Managed Insight Collections

To add insights to the `InsightManager` , return a list of `Insight` objects from the Update method of your Alpha model or call the `EmitInsights` method. If you manage an `InsightCollection` that's seperate from the `InsightManager` , there are some methods to add `Insight` objects to it.

To add an insight to an `InsightCollection` , call the `Add` method.

```
// You can manually add an Insight to an InsightCollection object to organize, fetch, or cancel insights an alpha has
generated.
_insightCollection.Add(insight);
```

To add a list of insights, call the `AddRange` method.

```
// You can manually add a list of Insights to an InsightCollection object to organize, fetch, or cancel insights an
alpha has generated.
_insightCollection.AddRange(insights);
```

Preserve Insights Between Deployments

Follow these steps to use the Object Store to preserve the algorithm state across live deployments:

1. Create an algorithm that defines a storage key and adds insights to the Insight Manager .

   ```
   public class ObjectStoreChartingAlgorithm : QCAlgorithm
   {
       private string _insightKey;
       public override void Initialize()
       {
           _insightKey = $"{ProjectId}/insights";
           SetUniverseSelection(new ManualUniverseSelectionModel(QuantConnect.Symbol.Create("SPY",
   SecurityType.Equity, Market.USA)));
           SetAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(5), 0.025,
   null));
       }
   }
   ```

2. In the OnEndOfAlgorithm event handler of the algorithm, get the Insight objects and save them in the Object Store as a JSON object.

```
C#
public override void OnEndOfAlgorithm()
{
    var insights = Insights.GetInsights(x => x.IsActive(UtcTime));
    ObjectStore.SaveJson(_insightKey, insights);
}
```

3. At the bottom of the `Initialize` method, read the Insight objects from the Object Store and add them to the Insight Manager .

```
C#
if (ObjectStore.ContainsKey(_insightKey))
{
    var insights = ObjectStore.ReadJson<List<Insight>>(_insightKey);
    Insights.AddRange(insights);
}
```

The following algorithm provides a full example of preserving the Insight state between deployments:

```
C#
public class ObjectStoreInsightsAlgorithm : QCAlgorithm
{
    private string _insightsKey = "insights";

    public override void Initialize()
    {
        UniverseSettings.Resolution = Resolution.Daily;

        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        _insightsKey = $"{ProjectId}/{_insightsKey}";

        // Read the file with the insights
        if (ObjectStore.ContainsKey(_insightsKey))
        {
            // Load cached JSON key value pair from object store, then add to algorithm insights.
            var insights = ObjectStore.ReadJson<List<Insight>>(_insightsKey);
            Log($"Read {insights.Count} insight(s) from the Object Store");
            Insights.AddRange(insights);

            // Delete the key to reuse it
            ObjectStore.Delete(_insightsKey);
        }

        SetUniverseSelection(new ManualUniverseSelectionModel(QuantConnect.Symbol.Create("SPY", SecurityType.Equity,
Market.USA)));
        SetAlpha(new ConstantAlphaModel(InsightType.Price, InsightDirection.Up, TimeSpan.FromDays(5), 0.025, null));
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel(Resolution.Daily));
    }

    public override void OnEndOfAlgorithm()
    {
        // Get all active insights.
        var insights = Insights.GetInsights(x => x.IsActive(UtcTime));
        // If we want to save all insights (expired and active), we can use.
        // var insights = Insights.GetInsights(x => true);
        Log($"Save {insights.Count} insight(s) to the Object Store.");
        ObjectStore.SaveJson(_insightsKey, insights);
    }
}
```

Properties

The `InsightManager` has the following properties:

Examples

The following examples demonstrate some common practices for using insight manager.

Example 1: Cancel Insight

If the risk management model only submits the `PortfolioTarget` to liquidate the security, it will be re-ordered immediately due to the active insight. Hence, we also need to add logic to cancel the insight in the `ManageRisk` method of the risk management model. The below example only permits 1-day holding time per each insight through a custom risk management model.

```C#
public class FrameworkInsightManagerAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 11, 15);
        SetCash(1000000);
        // Seed the price of each asset with its last known price to avoid trading errors.
        Settings.SeedInitialPrices = true;
        // Add a universe of the most liquid stocks since their trend is more capital-supported.
        AddUniverseSelection(new QC500UniverseSelectionModel());
        // Emit insights on EMA cross, indicating the trend changes. We use short-term versus medium-term for more
trade opportunities.
        AddAlpha(new EmaCrossAlphaModel(20, 60, Resolution.Daily));
        // Equal weighting on each insight is needed to dissipate capital risk evenly.
        SetPortfolioConstruction(new EqualWeightingPortfolioConstructionModel());

        // Add a risk management model that controls maximum holding time.
        AddRiskManagement(new MaximumHoldingTimeRiskManagementModel(TimeSpan.FromDays(5)));
    }

    private class MaximumHoldingTimeRiskManagementModel : RiskManagementModel
    {
        private TimeSpan _maximumHoldingTime;
        // A dictionary to hold the holding time of each position.
        private Dictionary<Symbol, DateTime> _holdingTime = new();

        public MaximumHoldingTimeRiskManagementModel(TimeSpan maximumHoldingTime)
        {
            _maximumHoldingTime = maximumHoldingTime;
        }

        public override IEnumerable<IPortfolioTarget> ManageRisk(QCAlgorithm algorithm, IPortfolioTarget[] _)
        {
            var targets = new List<PortfolioTarget>();

            foreach (var (symbol, holding) in algorithm.Portfolio)
            {
                if (holding.Invested)
                {
                    if (!_holdingTime.TryGetValue(symbol, out var time))
                    {
                        // For newly open positions, save the entry time.
                        _holdingTime[symbol] = time = algorithm.Time;
                    }
                    else if (algorithm.Time - time > _maximumHoldingTime)
                    {
                        // Liquidate the position if the holding time exceeds the permitted level.
                        targets.Add(new PortfolioTarget(symbol, 0m));
                        // Cancel the insight to avoid repeated re-ordering.
                        algorithm.Insights.Cancel(new[] { symbol });
                        // Remove from dictionary.
                        _holdingTime.Remove(symbol);
                    }
                }
            }

            return targets;
        }
    }
}
```

Charting

Introduction

We provide a powerful charting API that you can use to build many chart types.

Charts

Charts contain a collection of series, which display data on the chart. To add a chart to an algorithm, create a `Chart` object and then call the `AddChart` method.

```C#
var chart = new Chart("<chartName>");
AddChart(chart);
```

The `Chart` constructor expects a name argument. The following chart names are reserved:

- Assets Sales Volume
- Exposure
- Portfolio Margin

Series

A chart series displays data on the chart. To add a series to a chart, create a `Series` object and then call the `AddSeries` method.

```C#
var series = new Series("<seriesName>");
chart.AddSeries(series);
```

Arguments

There are several other headers for the `Series` constructor.

```
Series(name, type)
Series(name, type, index)
Series(name, type, index, unit)
Series(name, type, unit)
Series(name, type, unit, color)
Series(name, type, unit, color, symbol)
```

The following table describes the constructor arguments:

Argument	Data Type	Description
`name`	`string`	Name of the series
`type`	`SeriesType`	Type of the series
`index`	`int`	Index position on the chart of the series
`unit`	`string`	Unit for the series axis
`color`	`Color`	Color of the series
`symbol`	`ScatterMarkerSymbol`	Symbol for the marker in a scatter plot series

The default `Series` is a line chart with a "$" unit on index 0.

Names

The `Series` constructor expects a name argument. If you add a series to one of the default charts, some series names may be reserved. The following table shows the reserved series name for the default charts:

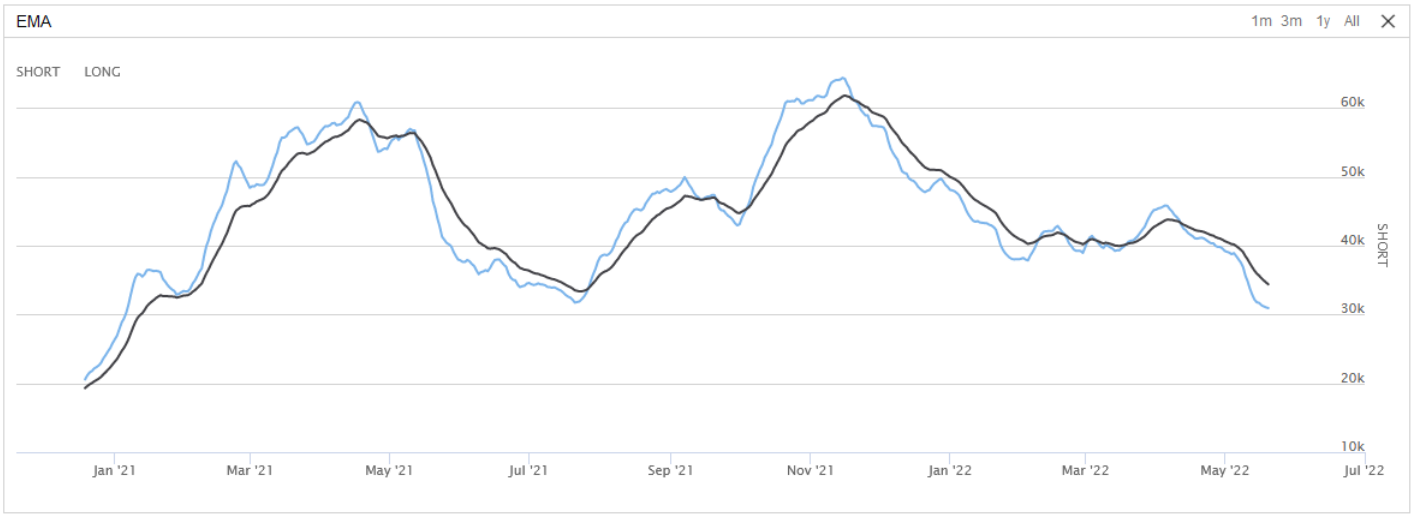
Chart Name	Reserved Series Names
Strategy Equity	Equity, Return
Capacity	Strategy Capacity
Drawdown	Equity Drawdown
Benchmark	Benchmark
Portfolio Turnover	Portfolio Turnover

Types

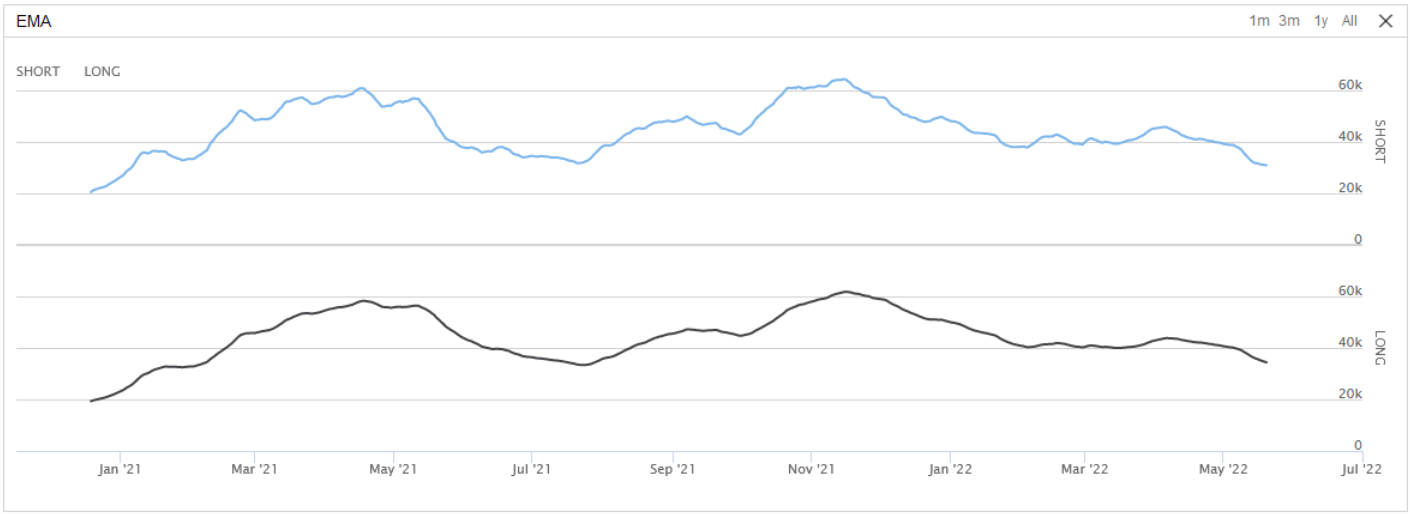
The `SeriesType` enumeration has the following members:

Index

The series index refers to its position in the chart. If all the series are at index 0, they lay on top of each other. If each series has its own index, each series will be separate on the chart. The following image shows an EMA cross chart with both EMA series set to the same index:



The following image shows the same EMA series, but with the short EMA on index 0 and the long EMA on index 1:



Colors

To view the available `Color` options, see the Color Struct Properties in the .NET documentation.

Scatter Marker Symbols

The `ScatterMarkerSymbol` enumeration has the following members:

Candlestick Series

A chart candlestick series displays candlesticks on the chart. To add a candlestick series to a chart, create a `CandlestickSeries` object and then call the `AddSeries` method.

```C#
var candlestickSeries = new CandlestickSeries("<seriesName>");
chart.AddSeries(candlestickSeries);
```

There are several other headers for the `CandlestickSeries` constructor.

```
CandlestickSeries(name)
CandlestickSeries(name, index)
CandlestickSeries(name, index, unit)
CandlestickSeries(name, unit)
```

The following table describes the constructor arguments:

Argument	Data Type	Description
`name`	`string`	Name of the series
`index`	`int`	Index position on the chart of the series
`unit`	`string`	Unit for the series axis

The default `CandlestickSeries` has 0 index and "$" unit.

Plot Series

To add a data point to a chart series, call the `Plot` method. If you haven't already created a chart and series with the names you pass to the `Plot` method, the chart and/or series is automatically created.

```
C#
Plot("<chartName>", "<seriesName>", value);
```

The `value` argument can be an integer or decimal number. If the chart is a time series, the value is added to the chart using the algorithm time as the x-coordinate.

To plot the current value of indicators, call the `Plot` method. The method accepts up to four indicators.

```
C#
// In Initialize
var symbol = AddEquity("SPY");
var smaShort = SMA(symbol, 10);
var smaLong = SMA(symbol, 20);

// In OnData
Plot("<chartName>", smaShort, smaLong)
```

To plot all of the values of some indicators, in the `Initialize` method, call the `PlotIndicator` method. The method plots each indicator value as the indicator updates. The method accepts up to four indicators.

```
C#
var symbol = AddEquity("SPY");
var smaShort = SMA(symbol, 10);
var smaLong = SMA(symbol, 20);
PlotIndicator("<chartName>", smaShort, smaLong)
```

Plot Candlestick

To add a sample of open, high, low, and close values to a candlestick series, call the `Plot` method with the data points. If you haven't already created a chart and series with the names you pass to the `Plot` method, the chart and/or series is automatically created.

```
C#
Plot("<chartName>", "<seriesName>", open, high, low, close);
```

The `open` , `high` , `low` , and `close` arguments can be an integer for decimal number. If the chart is a time series, the values are added to the chart using the algorithm time as the x-coordinate.

To plot the current trade bar, call the `Plot` method with a `TradeBar` argument in the `OnData` method.

```
// In Initialize
var equity = AddEquity("SPY");
var forex = AddForex("EURUSD");

// In OnData
var tradeBar = slice.Bars["SPY"];
var collapsed = slice.QuoteBars["EURUSD"].Collapse();   // Collapses QuoteBar into TradeBar object

Plot("<chartName1>", "<seriesName>", tradeBar)
Plot("<chartName2>", "<seriesName>", collapsed)
```

To plot consolidated bars, call the `Plot` method with a `TradeBar` argument in the consolidation handler.

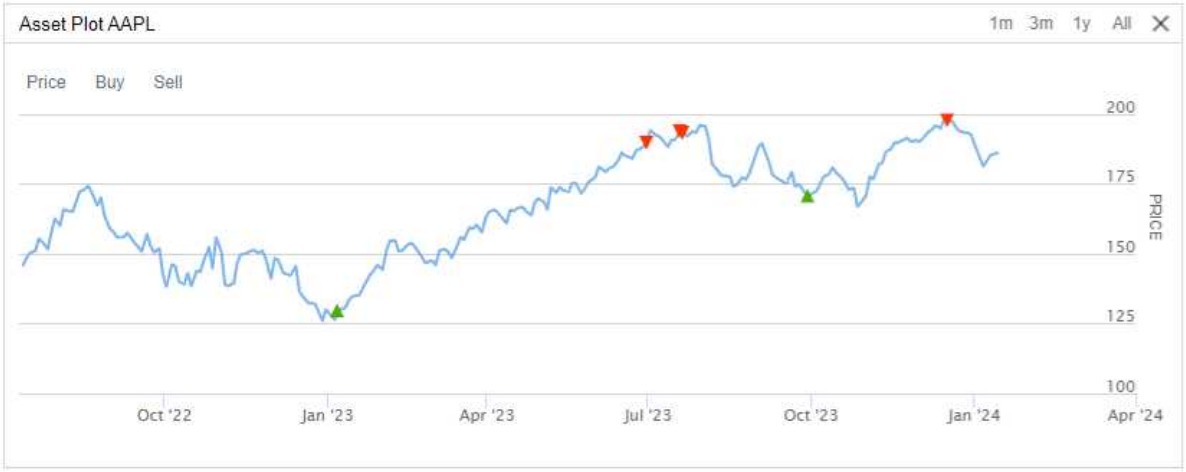
```
// In Initialize
var equity = AddEquity("SPY");
Consolidate(equity.Symbol, TimeSpan.FromMinutes(10), ConsolidationHandler);

// Define the consolidation handler
void ConsolidationHandler(TradeBar consolidatedBar)
{
    Plot("<chartName>", "<seriesName>", consolidatedBar)
}
```

Plot Asset Data

Asset plots display the trade prices of an asset and the following order events you have for the asset:

Order Event	Icon
Submissions	Gray circle
Updates	Blue circle
Cancellations	Gray square
Fills and partial fills	Green (buys) or red (sells) arrows

The following image shows an example asset plot for AAPL:



The order submission icons aren't visible by default.

For more information about these charts, including how to view them in QC Cloud, see Asset Plots for backtests or live trading .

View Charts

The following table describes where you can access your charts, depending on how to deploy your algorithms:

Location	Algorithm Lab Algorithms	CLI Cloud Algorithms	CLI Local Algorithms
Backtest results page	✓	✓	
Live results page	✓	✓	
/backtests/read endpoint	✓	✓	
/live/read endpoint	✓	✓	
ReadBacktest method	✓	✓	
ReadLiveAlgorithm method	✓	✓	
Local JSON file in your <projectName> / backtests / <timestamp> or <projectName> / live / <timestamp> directory		✓	✓

Quotas

When you run backtests, you must stay within the plotting quotas to avoid errors.

Cloud Quotas

Intensive charting requires hundreds of megabytes of data, which is too much to stream online or display in a web browser. The number of series and the number of data points per series you can plot depends on your organization tier . The following table shows the quotas:

Tier	Max Series	Max Data Points per Series
Free	10	4,000
Quant Researcher	10	8,000
Team	25	16,000
Trading Firm	25	32,000
Institution	100	96,000

If you exceed the series quota, your algorithm stops executing and the following message displays:

Exceeded maximum chart series count, new series will be ignored. Limit is currently set at <quota>.

If you exceed the data points per series quota, the following message displays:

Exceeded maximum points per chart, data skipped

If your plotting needs exceed the preceding quotas, create the plots in the Research Environment instead .

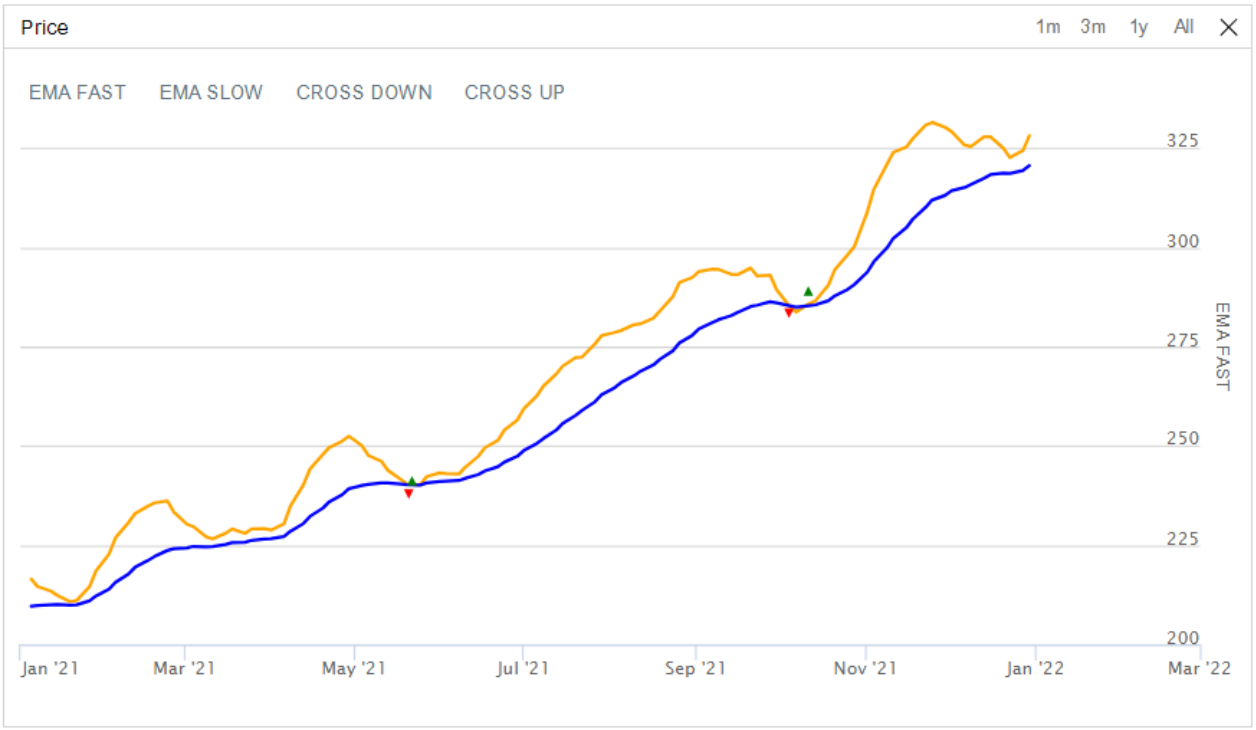
Local Quotas

If you execute local backtests, the charting quotas are set by the `maximum-chart-series` and `maximum-data-points-per-chart-series` configuration settings .

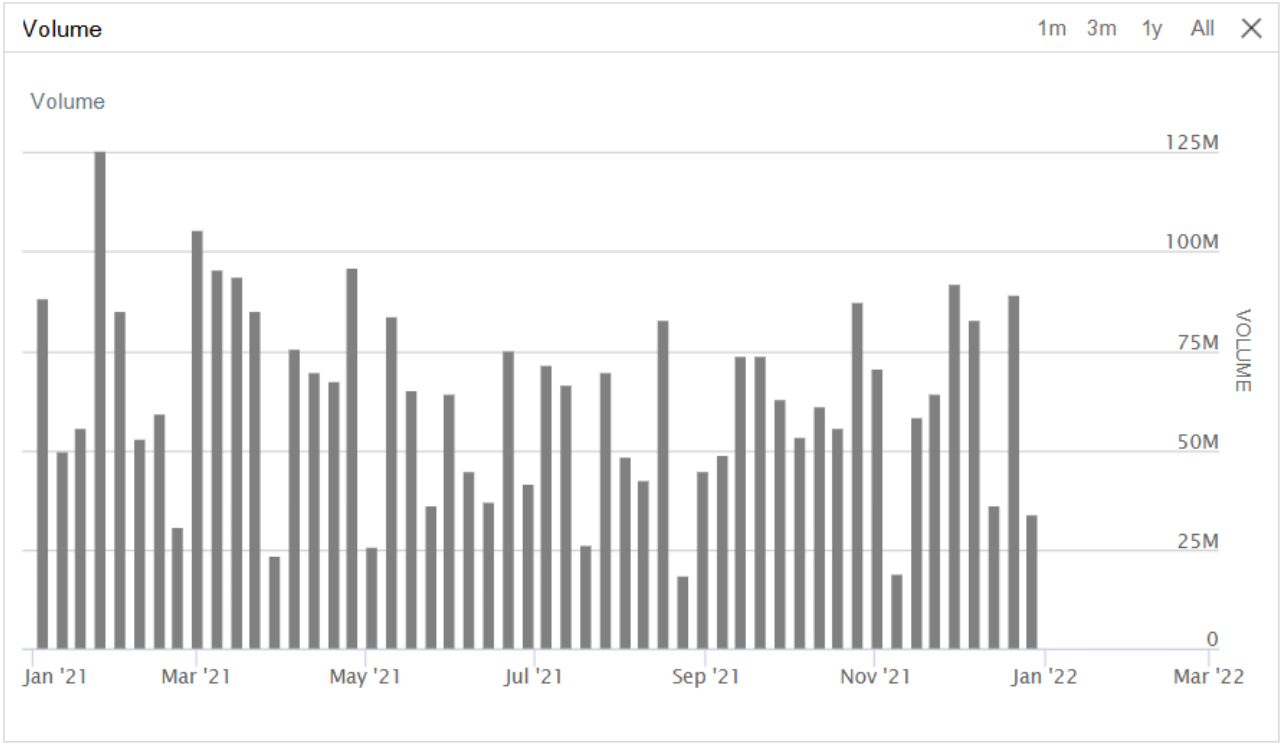
Examples

The following example shows how to plot the daily closing price of SPY with a scatter plot:

```C#
public class ChartingDemoAlgorithm : QCAlgorithm
{
    private ExponentialMovingAverage _emaFast;
    private ExponentialMovingAverage _emaSlow;
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        Settings.AutomaticIndicatorWarmUp = true;
        var symbol = AddEquity("MSFT", Resolution.Daily).Symbol;
        _emaFast = EMA(symbol, 10);
        _emaSlow = EMA(symbol, 50);
        var chart = new Chart("Price");
        AddChart(chart);
        chart.AddSeries(new Series("CROSS UP", SeriesType.Scatter, "$", Color.Green, ScatterMarkerSymbol.Triangle));
        chart.AddSeries(new Series("CROSS DOWN", SeriesType.Scatter, "$", Color.Red,
ScatterMarkerSymbol.TriangleDown));
        chart.AddSeries(new Series("EMA FAST", SeriesType.Line, "$", Color.Orange));
        chart.AddSeries(new Series("EMA SLOW", SeriesType.Line, "$", Color.Blue));
        chart = new Chart("Volume");
        AddChart(chart);
        chart.AddSeries(new Series("Volume", SeriesType.Bar, "", Color.Gray));
        chart = new Chart("Candlestick");
        AddChart(chart);
        chart.AddSeries(new CandlestickSeries("MSFT", "$"));
    }

    public override void OnEndOfDay(Symbol symbol)
    {
        var data = (TradeBar)Securities[symbol].GetLastData();
        if (data != null)
        {
            Plot("Candlestick", "MSFT", data);
            Plot("Volume", "Volume", data.Volume);
        }
        Plot("Price", "EMA FAST", _emaFast);
        Plot("Price", "EMA SLOW", _emaSlow);
        if (_emaFast > _emaSlow && _emaFast[1] < _emaSlow[1])
            Plot("Price", "CROSS UP", Securities[symbol].Price);
        else if (_emaFast < _emaSlow && _emaFast[1] > _emaSlow[1])
            Plot("Price", "CROSS DOWN", Securities[symbol].Price);
    }
}
```


Price
1m 3m 1y All
EMA FAST
EMA SLOW
CROSS DOWN
CROSS UP
325
300
275
250
225
200
EMA FAST
Jan '21
Mar '21
May '21
Jul '21
Sep '21
Nov '21
Jan '22
Mar '22

Candlestick
1m 3m 1y All
MSFT
350
325
300
275
250
225
200
175
MSFT
Jan '21
Mar '21
May '21
Jul '21
Sep '21
Nov '21
Jan '22
Mar '22



To see a full example, run the CustomChartingAlgorithm .

Logging

Introduction

Algorithms can record string messages ('log statements') to a file for analysis after a backtest is complete, or as a live algorithm is running. These records can assist in debugging logical flow errors in the project code. Consider adding them in the code block of an `if` statement to signify an error has been caught.

It's good practice to add logging statements to live algorithms so you can understand its behavior and keep records to compare against backtest results. If you don't add logging statements to a live algorithm and the algorithm doesn't trade as you expect, it's difficult to evaluate the underlying problem.

Log Messages

Log statements are added to the log file while your algorithm continues executing. Logging dataset information is not permitted. Use `Log` method statements to debug your backtests and live trading algorithms.

If you execute algorithms in QuantConnect Cloud, log length is capped by organization tier . If your organization hits the daily limit, contact us .

If you log the same content multiple times, only the first instance is added to the log file. To bypass this rate-limit, add a timestamp to your log messages.

For live trading, the log files of each cloud project can store up to 100,000 lines for up to one year. If you log more than 100,000 lines or some lines become older than one year, we remove the oldest lines in the files so your project stays within the quota.

To record the algorithm state when the algorithm stops executing, add log statements to the `OnEndOfAlgorithm` event handler.

```
C#
Log("My log message");
```

Debug Messages

Debug statements are the same as log statements, but `Debug` method statements are orange in the Cloud Terminal. Use these statements when you want to give more attention to a message in the Cloud Terminal. Debug messages can be up to 200 characters in length. If you send multiple debug statements within 1 second, your messages are rate-limited to avoid crashing your browser.

```
C#
Debug("My debug message");
```

Error Messages

Error statements are the same as log statements, but `Error` method statements are displayed in red text in the Cloud Terminal. Use these statements when you want to give the most attention to a message in the Cloud Terminal. Error statements are rate-limited like debug statements.

```
Error("My error message");
```

Quit Messages

Quit statements cause your project to stop running and may log some data to the log file and Cloud Terminal. These statements are orange in the Cloud Terminal. When you call the `Quit` method method, the program continues executing until the end of the method definition. If you want to quit execution immediately, `return` after you call `Quit` method.

```
Quit("My quit message");
```

Get Logs

The following tables describe how to access the logs of your backtests and live algorithms:

Deployment Target	Execution Mode	Access Tools
QuantConnect Cloud	Backtest	• Backtest results page in the Algorithm Lab • Backtest results page on Local Platform
QuantConnect Cloud	Live Trading	• Live results page in the Algorithm Lab • Live results page on Local Platform • /live/read/log endpoint
Local	Backtest	• Backtest results page on Local Platform • <organizationWorkspace> / <projectName> / backtests / <unixTimestamp> / <algorithmId>-log.txt • lean logs CLI command
Local	Live Trading	• Live results page on Local Platform • <organizationWorkspace> / <projectName> / live / <timeStamp> / / L-<deploymentId>-log.txt • lean logs CLI command

Example

The following examples demonstrate some common practices for logging.

Example 1: On Events And Trading Logic

The following example trades a simple EMA cross trend-following strategy on SPY. We log or debug messages in event handlers and trading logic changes to get information on any event flow.

C#

```
public class LoggingAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private ExponentialMovingAverage _ema;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 12);
        SetEndDate(2024, 10, 1);
        SetCash(1000000);

        // Request SPY data to trade.
        _spy = AddEquity("SPY").Symbol;
        // Create an EMA indicator to generate trade signals.
        _ema = EMA(_spy, 20, Resolution.Daily);

        // Warm up the algorithm for indicator readiness.
        SetWarmUp(20, Resolution.Daily);
    }

    public override void OnWarmupFinished()
    {
        // Signals on warm-up finished.
        Log("Warm up finished");
    }

    public override void OnData(Slice slice)
    {
        if (!IsWarmingUp && slice.Bars.TryGetValue(_spy, out var bar))
        {
            // Trend-following strategy using price and EMA.
            // If the price is above EMA, SPY is in an uptrend, and we buy it.
            if (bar.Close > _ema && Portfolio[_spy].IsLong)
            {
                Debug("Trend changes to upwards");
                SetHoldings(_spy, 1m);
            }
            else if (bar.Close < _ema && Portfolio[_spy].IsShort)
            {
                Debug("Trend changes to downwards");
                SetHoldings(_spy, -1m);
            }
        }
    }

    public override void OnOrderEvent(OrderEvent orderEvent)
    {
        if (orderEvent.Status == OrderStatus.Filled)
        {
            // Log the order details if being filled.
            Log($"Order filled - Quantity: {orderEvent.Quantity}");
        }
    }
}
```

Statistics

LEAN automatically monitors a set of statistics as your algorithms execute. You can even create custom statistics for further analysis.

Algorithm Statistics

Trade statistics and rolling KPIs

Runtime Statistics

Performace at a single moment in time

Trade Statistics

Get and customize trade statistics

See Also

Cloud Backtest Results

Statistics

Algorithm Statistics

Introduction

The `StatisticsResults` object tracks all the statistics of your algorithm. You can access the statistics anywhere in your algorithm where you have a reference to the algorithm class. Some of these statistics are a function of the risk free interest rate and the number of trading days per year .

Properties

The `StatisticsResults` object has the following properties:

Add Statistics

To add custom statistics to the `StatisticsResults` object, call the `SetSummaryStatistic` method with the statistic name and value.

```
C#
SetSummaryStatistic("name", value);
```

Get Values

To get the `StatisticsResults` , use the `Statistics` member of the algorithm class.

```
C#
var sharpe = Statistics.TotalPerformance.PortfolioStatistics.SharpeRatio;
```

To get the value of a custom statistic while your algorithm runs, index the `Summary` member with the statistic name. The values in the `Summary` dictionary are strings, so you may need to cast the value that returns to a different data type.

```
C#
var value = Statistics.Summary["name"];
```

The following table describe other ways to get the statistics after your backtest completes:

Deployment Target	Access Tools
QuantConnect Cloud	• Backtest results page in the Algorithm Lab • Backtest results page on Local Platform • ReadBacktest API method • /backtests/read endpoint
Local	• Backtest results page on Local Platform • <organizationWorkspace> / <projectName> / backtests / <unixTimestamp> / <algorithmId>.json

Examples

The following examples demonstrate common practices for getting algorithm statistics.

Example 1: Portfolio Statistics

To evaluate the algorithm's best domain or regime, we can log the portfolio statistics daily along the backtest to analyze. To do so, we log the Sharpe Ratio, Beta, and 95% Value-at-Risk at the end of each day, storing them in the object store for assessment later.

C#

```
public class AlgorithmStatisticsAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private ExponentialMovingAverage _ema;
    private List<string> _statistics = new() { "Time,Sharpe,Beta,95VaR" };

    public override void Initialize()
    {
        SetStartDate(2024, 9, 12);
        SetEndDate(2024, 10, 1);
        SetCash(1000000);

        // Request SPY data to trade.
        _spy = AddEquity("SPY").Symbol;
        // Create an EMA indicator to generate trade signals.
        _ema = EMA(_spy, 20, Resolution.Daily);
        // Warm-up indicator for immediate readiness.
        WarmUpIndicator(_spy, _ema, Resolution.Daily);
    }

    public override void OnData(Slice slice)
    {
        if (slice.Bars.TryGetValue(_spy, out var bar))
        {
            // Trend-following strategy using price above or below EMA.
            // If the price is above EMA, SPY is in an uptrend, and we buy it.
            if (bar.Close > _ema && !Portfolio[_spy].IsLong)
            {
                SetHoldings(_spy, 1m);
            }
            else if (bar.Close < _ema && !Portfolio[_spy].IsShort)
            {
                SetHoldings(_spy, -1m);
            }
        }
    }

    public override void OnEndOfDay(Symbol symbol)
    {
        // Obtain the algorithm statistics interested.
        var sharpe = Statistics.TotalPerformance.PortfolioStatistics.SharpeRatio;
        var b = Statistics.TotalPerformance.PortfolioStatistics.Beta;
        var var = Statistics.TotalPerformance.PortfolioStatistics.ValueAtRisk95;

        // Plot the statistics.
        Plot("Statistics", "Sharpe", sharpe);
        Plot("Statistics", "Beta", b);
        Plot("Statistics", "Value-at-Risk", var);

        // Write to save the statistics.
        _statistics.Add($"{Time:yyyyMMdd},{sharpe},{b},{var}");
    }

    public override void OnEndOfAlgorithm()
    {
        // Save the logged statistics for later access in the object store.
        ObjectStore.Save($"{ProjectId}/algorithm-statistics", string.Join('\n', _statistics));
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithms
StatisticsResultsAlgorithm.cs C#

Statistics

Runtime Statistics

Introduction

Runtime statistics show the performace of your algorithm at a single moment in time. Some of these statistics are a function of the risk free interest rate and the number of trading days per year .

Default Statistics

The following table describes the default runtime statistics:

Statistic	Description
Equity	The total portfolio value if all of the holdings were sold at current market rates.
Fees	The total quantity of fees paid for all the transactions.
Holdings	The absolute sum of the items in the portfolio.
Net Profit	The dollar-value return across the entire trading period.
PSR	The probability that the estimated Sharpe ratio of an algorithm is greater than a benchmark (1).
Return	The rate of return across the entire trading period.
Unrealized	The amount of profit a portfolio would capture if it liquidated all open positions and paid the fees for transacting and crossing the spread.
Volume	The total value of assets traded for all of an algorithm's transactions.

Add Statistics

To add a custom runtime statistic, call the `SetRuntimeStatistic` method with a `name` and `value` . The `value` argument can be a `string` or a number.

```
SetRuntimeStatistic(name, value);
```

Don't try to set a value for any of the preceding default runtime statistics. LEAN overwrites the value that you try to set.

Get Values

To get the value of a runtime statistic in an algorithm, index the `RuntimeStatistics` member of the algorithm class with the statistic name. The values of the `RuntimeStatistics` dictionary are strings, so you may need to cast the result to a different data type.

```
var value = RuntimeStatistics[name];
```

The following table describe other ways to get the runtime statistics of your backtests and live algorithms:

Deployment Target	Execution Mode	Access Tools
QuantConnect Cloud	Backtest	• Backtest results page in the Algorithm Lab • Backtest results page on Local Platform • ReadBacktest API method • /backtests/read endpoint
QuantConnect Cloud	Live Trading	• Live results page in the Algorithm Lab • Live results page on Local Platform • ReadLiveAlgorithm API method • /live/read endpoint
Local	Backtest	• Backtest results page on Local Platform • <organizationWorkspace> / <projectName> / backtests / <unixTimestamp> / <algorithmId>.json
Local	Live Trading	• Live results page on Local Platform • <organizationWorkspace> / <projectName> / live / <timeStamp> / / L-<deploymentId>.json

Examples

The following examples demonstrate common practices for setting runtime statistics.

Example 1: Display Banner

The following algorithm demonstrates how to put calculated runtime statistics of interest in the top display banner in the backtest interface and plot the data for analysis.

C#

```
public class RuntimeStatisticsAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private ExponentialMovingAverage _ema;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 12);
        SetEndDate(2024, 10, 1);
        SetCash(1000000);

        // Request SPY data to trade.
        _spy = AddEquity("SPY").Symbol;
        // Create an EMA indicator to generate trade signals.
        _ema = EMA(_spy, 20, Resolution.Daily);
        // Warms up indicator for immediate readiness.
        WarmUpIndicator(_spy, _ema, Resolution.Daily);
    }

    public override void OnData(Slice slice)
    {
        if (slice.Bars.TryGetValue(_spy, out var bar))
        {
            // Trend-following strategy using price and EMA.
            // If the price is above the EMA, SPY is in an uptrend, and we buy it.
            if (bar.Close > _ema && !Portfolio[_spy].IsLong)
            {
                SetHoldings(_spy, 1m);
            }
            else if (bar.Close < _ema && !Portfolio[_spy].IsShort)
            {
                SetHoldings(_spy, -1m);
            }
        }
    }

    public override void OnEndOfDay(Symbol symbol)
    {
        // Set the 95VaR and fee-to-profit ratio to the display banner.
        var var = Statistics.TotalPerformance.PortfolioStatistics.ValueAtRisk95;
        var feeToProfit = Statistics.TotalPerformance.TradeStatistics.TotalProfitLoss != 0 ?
            Statistics.TotalPerformance.TradeStatistics.TotalFees /
Statistics.TotalPerformance.TradeStatistics.TotalProfitLoss :
            0m;
        SetRuntimeStatistic("95VaR", var);
        SetRuntimeStatistic("Fee-to-Profit", feeToProfit);

        // Plot the calculated statistics.
        Plot("Statistics", "95VaR", var);
        Plot("Statistics", "fee-to-profit", feeToProfit);
    }

    public override void OnEndOfAlgorithm()
    {
        // Log the 95VaR when the algorithm stops
        var valueAtRisk95 = RuntimeStatistics["95VaR"];
        Log($"Final valueAtRisk95={valueAtRisk95}");
    }
}
```

Statistics

Trade Statistics

Introduction

The `TradeBuilder` tracks the trades of your algorithm and calculates some statistics. Some of these statistics are a function of the risk free interest rate and the number of trading days per year . You can adjust how the `TradeBuilder` defines a trade and how it matches offsetting order fills.

Set Trade Builder

To set the `TradeBuilder` , in the `Initialize` method, call the `SetTradeBuilder` method.

```
C#
SetTradeBuilder(new TradeBuilder(groupingMethod, matchingMethod));
```

The following table describes the arguments the `TradeBuilder` constructor accepts:

Argument	Data Type	Description	Default Value
`groupingMethod`	`FillGroupingMethod`	The method used to group order fills into trades	
`matchingMethod`	`FillMatchingMethod`	The method used to match offsetting order fills	

The `FillGroupingMethod` enumeration has the following members:

The `FillMatchingMethod` enumeration has the following members:

Default Behavior

The default `TradeBuilder` uses `FillGroupingMethod.FillToFill` and `FillMatchingMethod.FIFO` .

Check Open Positions

To check if you have a position open for a security as defined by the `FillGroupingMethod` , call the `HasOpenPosition` method.

```
C#
// Check if the trade builder has recorded any active trades (unclosed positions)
// for a security.
var hasOpenPosition = TradeBuilder.HasOpenPosition(_symbol);
```

Get Closed Trades

To get your closed trades, use the `ClosedTrades` property.

```
C#
var trades = TradeBuilder.ClosedTrades;
```

This property returns a list of `Trade` objects, which have the following attributes:

Examples

The following examples demonstrate common practices for using trade statistics.

Example 1: Kelly Criterion Sizing Using Previous Trades

The following algorithm uses the previous trades to update the win probability and expected return of each trade of the EMA cross strategy of SPY, by the concept of Bayes Theorem. The updated figures are used for position sizing through Kelly Criterion.

C#

```
public class TradeStatisticsAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private ExponentialMovingAverage _ema;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 12);
        SetEndDate(2024, 10, 1);
        SetCash(1000000);

        // Request SPY data to trade.
        _spy = AddEquity("SPY").Symbol;
        // Create an EMA indicator to generate trade signals.
        _ema = EMA(_spy, 20, Resolution.Daily);
        // Warm-up indicator for immediate readiness.
        WarmUpIndicator(_spy, _ema, Resolution.Daily);

        // Set up a trade builder to track the trade statistics; we are interested in open-to-close round trade.
        SetTradeBuilder(new TradeBuilder(FillGroupingMethod.FlatToFlat, FillMatchingMethod.FIFO));
    }

    public override void OnData(Slice slice)
    {
        if (slice.Bars.TryGetValue(_spy, out var bar))
        {
            // Trend-following strategy using price and EMA.
            // If the price is above EMA, SPY is in an uptrend, and we buy it.
            var sign = 0m;
            if (bar.Close > _ema && !Portfolio[_spy].IsLong)
            {
                sign = 1m;
            }
            else if (bar.Close < _ema && !Portfolio[_spy].IsShort)
            {
                sign = -1m;
            }
            else
            {
                return;
            }

            var size = 1m;
            var trades = TradeBuilder.ClosedTrades;

            if (trades.Count > 4)
            {
                // Use the trade builder to obtain the win rate and % return of the last five trades to calculate
position size.
                var lastFiveTrades = trades.OrderBy(x => x.ExitTime).TakeLast(5);
                var probWin = lastFiveTrades.Count(x => x.IsWin) / 5m;
                var winSize = lastFiveTrades.Average(x => x.ProfitLoss / x.EntryPrice);
                // Use the Kelly Criterion to calculate the order size.
                size = probWin - (1 - probWin) / winSize;
            }

            SetHoldings(_spy, sign * size);
        }
    }
}
```

Live Trading

QuantConnect enables you to run your algorithms in live mode with real-time market data. You can use the same algorithm for backtesting and live trading with different brokerages and data providers.

Key Concepts

Synchonize your account and algorithm

Brokerages

Stream data into your algorithm

Data Providers

Send orders to your brokerage

Trading and Orders

Reconciliation

Differences from backtesting

Notifications

Stay informed on algorithm decisions

Commands

Display information on algorithm decisions

Arbitrage

Charting and Logging

Signal Exports

Send portfolio targets to third-party services

See Also

QuantConnect Live Trading
Supported Brokerages

Live Trading

Key Concepts

Introduction

Algorithms should generally work seamlessly shifting from backtesting to live trading with no changes required. In backtesting algorithm events are triggered as fast as possible, whereas in live trading events are triggered in realtime.

Algorithm Portfolio

The live trading algorithm portfolio is loaded from the brokerage holdings, including open orders. When there are unsupported asset types in the portfolio the live trading strategy is stopped as we cannot be sure how to model the asset.

In paper trading the algorithm starts from no portfolio, and attempts to remember the previous portfolio state from the last time the portfolio was deployed.

Your algorithm should not assume it starts from an empty portfolio as you may have holdings from previous deployments.

Stateful Strategies

There is no automatic state management of live strategies in QuantConnect. We recommend algorithms reconstruct state or indicator needs using the WarmUp and History methods.

Once the algorithm has warmed up some objects you can use the Object Store to save the data for the next algorithm restart.

In the event the algorithm is terminated unexpectedly, you should review the live algorithm portfolio at the brokerage to confirm it will behave as expected.

Data Delivery Times

Most data is available at the same time as in backtesting, with the exception of daily data which is often delayed by 4-8 hours (4am - 8am).

US Equities universe selection data is processed at 7-8am ET.

Latency

Live data takes time to travel from the source to your algorithm. The QuantConnect latencies vary depending on the data provider, but for US Equities, we have a latency of 5-40 milliseconds. A much more significant source of latency is the round trip order times from brokers, which can vary from 100ms to 5 seconds. QuantConnect is not intended for high-frequency trading, but we have integrations to high-speed brokers if you need.

Other Brokerages

If we do not support your brokerage reach out to let us and we can attempt to add them to the roadmap. With enough community requests we can build and open-source the brokerage integration.

Live Trading

Brokerages

Introduction

Brokerages provide you with a connection to the market so you can fill trades. To avoid placing invalid orders for execution in live trading, LEAN validates your orders before sending them to the real brokerage. To view all of the integrated brokerages, see Brokerages .

Portfolio

In live trading, LEAN populates the `Portfolio` object with your account holdings and the `Transactions` object with your open positions. If you don't manually subscribe to the assets in your account, LEAN subscribes to them with the lowest resolution of the subscriptions in your algorithm. For example, say you hold AAPL shares in your account and create the following subscriptions in your algorithm:

```
C#
AddEquity("SPY", Resolution.Hour);
AddEquity("MSFT", Resolution.Second);
```

In this case, LEAN subscribes to second-resolution data for AAPL since the lowest resolution in your algorithm is `Resolution.Second` .

Deposits and Withdrawals

You can deposit and withdraw cash from your brokerage account while you run an algorithm that's connected to the account. We sync the algorithm's cash holdings with the cash holdings in your brokerage account every day at 7:45 AM Eastern Time (ET).

Monitor the Brokerage Connection

We notify your algorithm when your brokerage connection disconnects and reconnects.

Lost Connections

If the brokerage connection breaks, we notify your algorithm through the `OnBrokerageDisconnect` event handler.

```
C#
public override void OnBrokerageDisconnect()
{
    Debug("Brokerage connection lost");
}
```

Restored Connections

When the brokerage connection restores after it disconnects, we notify your algorithm through the `OnBrokerageReconnect` event handler.

```
public override void OnBrokerageReconnect()
{
    Debug("Brokerage connection restored");
}
```

When LEAN reconnects with your brokerage, it synchronizes the state of your live orders. For example, say the brokerage fills your limit order during the period of disconnection. When the connection restores, LEAN updates the state of your portfolio and orders to reflect the filled order, but you won't receive an order event .

Example Algorithm

For a full example algorithm that implements the `OnBrokerageDisconnect` and `OnBrokerageReconnect` methods, see the BrokerageActivityEventHandlingAlgorithm in the LEAN GitHub repository.

Monitor Brokerage Messages

When your brokerage sends you a message, we notify your algorithm through the `OnBrokerageMessage` event handler.

```
public override void OnBrokerageMessage(BrokerageMessageEvent messageEvent)
{
    Debug(f"Brokerage message received: {messageEvent.Message}");
}
```

`BrokerageMessageEvent` objects have the following attributes:

For a full example algorithm that implements the `OnBrokerageMessage` method, see the BrokerageActivityEventHandlingAlgorithm in the LEAN GitHub repository.

To handle brokerage messages outside of your algorithm class, create and set a `BrokerageMessageHandler` . For a more information example, see the Reality Modeling > Brokerage Message Handler .

Brokerage Models

Brokerage features and limitations are realistically modelled with Brokerage Models. These reality models set the supported order and asset types for the brokerage. To learn how to set your specific brokerage, see Supported Models .

Examples

The following examples demonstrate common practices for handling live brokerage messages.

Example 1: Pre-Open Order Error

The following algorithm simulate extended market hour trading on Tradier Brokerage . We place a market-on-open order at 7:30am and expect the order cannot submit to the broker since it is not supported, resulting into an error.

C#

```
public class OrderErrorsAlgorithm : QCAlgorithm
{
    private Symbol _spy;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 5);
        // Simulate Tradier brokerage, which does not support the market-on-open orders.
        SetBrokerageModel(BrokerageName.TradierBrokerage, AccountType.Cash);

        // Request extended market hour SPY data for trading.
        _spy = AddEquity("SPY", extendedMarketHours: true).Symbol;

        // Set a scheduled event to trade 2 hours pre-open to place the market-on-open-order.
        Schedule.On(
            DateRules.EveryDay(_spy),
            TimeRules.BeforeMarketOpen(_spy, 120),
            OpenPosition
        );
    }

    private void OpenPosition()
    {
        // Buy on market open will result in an error since Tradier does not support it.
        MarketOnOpenOrder(_spy, 10);
    }

    // It will be triggered on live trading.
    public override void OnBrokerageMessage(BrokerageMessageEvent message)
    {
        if (message.Type == BrokerageMessageType.Error)
        {
            Log($"{Time}: {message.Type}: Message: {message.Message}");
        }
    }
}
```

Live Trading

Data Providers

Introduction

Data providers provide a stream of asset prices and quotes delivered to your trading algorithm during live execution. You need live data provider to inject data into your algorithm so that you can make real-time trading decisions and so that the values of the securities in your portfolio update in real-time. You can source data from QuantConnect, brokerage, or a third-party source. To view the available data providers you can use in QuantConnect cloud, see Datasets . To view the available data providers you can use with local algorithms, see the Data Providers .

Data Providers

You can source data from QuantConnect, your brokerage, or third-party providers. The following table lists the data providers:

Data Provider	Cloud	Local
QuantConnect	✓	
Alpaca	✓	✓
Alpha Vantage		✓
Binance	✓	✓
Bitfinex	✓	✓
Bybit	✓	✓
Charles Schwab	✓	✓
Coinbase	✓	✓
FactSet		✓
Interactive Brokers	✓	✓
IQFeed		✓
Kraken	✓	✓
Polygon	✓	✓
Samco	✓	✓
Theta Data		✓
TradeStation	✓	✓
Terminal Link		✓
Tradier	✓	✓
Zerodha	✓	✓

Bar Building

We aggregate ticks to build bars.

In live trading, bars are built using the exchange timestamps with microsecond accuracy. This microsecond-by-microsecond processing of the ticks can mean that the individual bars between live trading and backtesting can have slightly different ticks. As a result, it's possible for a tick to be counted in different bars between backtesting and live trading, which can lead to bars having slightly different open, high, low, close, and volume values.

Latency

Live data takes time to travel from the source to your algorithm. The QuantConnect latencies vary depending on the data provider, but for US Equities, we have a latency of 5-40 milliseconds. A much more significant source of latency is the round trip order times from brokers, which can vary from 100ms to 5 seconds. QuantConnect is not intended for high-frequency trading, but we have integrations to high-speed brokers if you need.

History Requests

In live trading, if you make a history request for minute data at noon and the history period covers the start of the previous day to the present moment, the data from the previous day will be backtest data. The data of the current day will be live data that we collected throughout the morning. If you make this history request in a backtest, you might get slightly different data for the current day because of post-processing from the data vendor.

In cloud deployments, QuantConnect is the default data provider. If you remove it and use a different data provider, you won't get any price data from QuantConnect. However, we provide the following datasets:

- US Equity Security Master
- US Futures Security Master
- US Equity Option Universe
- US Index Option Universe
- Universe selection datasets
- Non-streaming alternative datasets

Warm Up Periods

LEAN supports an automated fast-forward system called "Warm Up". It simulates winding back the date you deployed your algorithm by a specific time period. In backtests, it changes the start date of your algorithm to an earlier time. In live trading, it fetches data from before your deployment and fast forwards through your strategy up to the present day. Warm Up is a great way to prepare your algorithm and its indicators for trading. You can't place trades during the warm-up period because the data feed is replaying historical data for setting algorithm state.

Live Trading

Trading and Orders

Introduction

LEAN has dozens of methods to create, update, and cancel orders. You can place orders automatically with helper methods or manually through methods on the algorithm API. You can fetch, update, and cancel manual orders with order tickets . As the state of your orders change, LEAN creates events that notify your algorithm.

In backtesting, LEAN simulates order fills with historical data, but you can create your own fill, fee, slippage, and margin models via plugin points. You control how optimistic or pessimistic order fills are with transaction model classes. For more information about these types of models, see Reality Modeling .

In live trading, orders fill asynchronously. We send your order to the API of your brokerage and wait for their response to update the state of your algorithm and portfolio. The timing of live order fills doesn't depend on the resolution of your security subscriptions. When your order fills, the fill price and fee is set by your brokerage. You can add event handlers to your algorithm to monitor the brokerage connection and brokerage messages .

In backtesting, the trade fill timing depends on the resolution of your security subscription. For example, if you subscribe to a security with minute resolution data, the algorithm only receives data in one-minute time slices . As a result, the fill model can only evaluate if the order should fill on a minute-by-minute frequency.

Order Life Cycle

When you call one of the order methods to place an order, LEAN performs pre-order checks to ensure that the order meets some requirements and uses the last known price to check you have enough capital . To see the requirements of each order, see the Requirements section of the documentation for the order types and the Orders section of your brokerage model . If you can place the order, LEAN creates `Order` and OrderTicket objects. The order ticket is sent to your brokerage. As the brokerage processes your order, it returns another order ticket and it's compared against the order to see if the order is satisfied. Orders fill asynchronously in live trading, so if you want to change an order, you must request it with the order ticket. Order changes are not guaranteed since your order may fill by the time brokerage receives the request.

When you create an order ticket, LEAN validates it to avoid sending orders that will be rejected by your brokerage. If the order is invalid, it's status is set to `OrderStatus.Invalid` . Otherwise, LEAN sets the order status to `OrderStatus.New` , sends the order to your brokerage, and waits for the brokerage message. If your brokerage accepts the order, the order status becomes `OrderStatus.Submitted` . Otherwise, it becomes `OrderStatus.Canceled` .

You can update orders until they fill or the brokerage prevents modifications. If you place an order update request, LEAN validates it to avoid sending orders that will be rejected by you brokerage. If the order update is valid, LEAN sends the update request and waits for the brokerage message. If your brokerage accepts the update, the order status becomes `OrderStatus.UpdateSubmitted` . Otherwise, it becomes `OrderStatus.Canceled` .

The brokerage notifies LEAN when you order fills (`OrderStatus.Filled`), partially fills (`OrderStatus.PartiallyFilled`) or is canceled (`OrderStatus.Canceled`). The brokerage can cancel your order before it completely fills. For example, if you place a market on open and the asset price moves before the market open to where you can't afford the quantity of the order, the brokerage rejects your order. The brokerage can also cancel your market on open order after partial fills if there is no shares left

to trade.

Transaction Manager

LEAN doesn't save orders and order tickets across live deployments.

When you deploy your algorithm, LEAN fetches the open orders from your brokerage and populate the algorithm transactions manager (`SecurityTransactionManager`).

Disable Buying Power

You can disable order margin checks and opt to let your brokerage decide to accept or reject the trades. This is helpful in live trading if you have a more permissive brokerage margin allowance that what LEAN models. The default position group buying power models are helpful for Option trading strategies. However, it can be counterproductive if it's not a supported Option strategy . To disable the validations of the default position group buying power model, use the `NullSecurityPositionGroupModel` . To set the `NullSecurityPositionGroupModel` for the portfolio, during initialization , call the `SetPositions` method with the `SecurityPositionGroupModel. Null` argument.

```
C#

Portfolio.SetPositions(SecurityPositionGroupModel.Null);
```

To disable the validations of the default buying power model , use the `NullBuyingPowerModel` . To set the `NullBuyingPowerModel` for a security subscription, call the `SetBuyingPowerModel` method with the `BuyingPowerModel. Null` argument.

```
C#

public override void Initialize()
{
    var equity = AddEquity("SPY");
    // To disable any buying power of the selected security
    equity.SetBuyingPowerModel(BuyingPowerModel.Null);
    // Alias:
    // equity.SetMarginModel(SecurityMarginModel.Null);
}
```

You can also set the `NullBuyingPowerModel` in a security initializer . If your algorithm has a universe, use the security initializer technique. To set the buying power of securities in the security initializer, set the brokerage model , set the security initializer , and then create security subscriptions.

```
C#

public class AddSecurityInitializerExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the security initializer to set models of assets.
        AddSecurityInitializer(CustomSecurityInitializer);
    }

    private void CustomSecurityInitializer(Security security)
    {
        // Overwrite the security buying power
        security.SetBuyingPowerModel(BuyingPowerModel.Null);
    }
}
```

Trading and Orders

Financial Advisors

Introduction

Financial Advisor accounts enable certified professionals to use a single trading algorithm to manage several client accounts. Our Interactive Brokers integration enables you to place FA group orders if your IB account code starts with F, FA, or I.

Group Routing

To place trades using a subset of client accounts, create Account Groups in Trader Workstation and then define the `InteractiveBrokersOrderProperties` when you create orders.

```
C#
DefaultOrderProperties = new InteractiveBrokersOrderProperties
{
    FaGroup = "TestGroupEQ",
    FaMethod = "Equal",
    Account = "DU123456"
};
```

SecurityHolding objects aggregate your positions across all the account groups. If you have two groups where group A has 10 shares of SPY and group B has -10 shares of SPY, then `Portfolio["SPY"].Quantity` is zero.

Allocation Methods

LEAN supports several allocation methods for FA group orders. If you intend to use the same group allocation method for every order, set the `DefaultOrderProperties` of your algorithm, which sets the order properties for all of your orders.

```
C#
public override void Initialize()
{
    // Set the default order properties
    DefaultOrderProperties = new InteractiveBrokersOrderProperties()
    {
        FaGroup = "TestGroupEQ",
        FaMethod = "Equal",
        Account = "DU123456"
    };
}

public override void OnData(Slice slice)
{
    // Use default order order properties
    LimitOrder(_symbol, quantity, limitPrice);
}
```

To adjust the order properties of an order, change the `DefaultOrderProperties` or pass an order properties object to the order method . The following sections explain the FA group allocation methods.

Equal

This group allocation method distributes shares equally between all accounts in the group. When you use this method, you need to specify an order quantity.

For example, say your Account Group includes four accounts and you place an order to buy 400 shares of a stock. In this case,

each account receives 100 shares. If your Account Group includes six accounts, each account receives 66 shares, and then 1 share is allocated to each account until all are distributed. After you submit the order, your algorithm receives order events to track the order progress. When all the shares are bought, the order status is `OrderStatus.Filled`. If one of the accounts in the group can't afford 10 of the shares it needs to buy, 10 shares are cancelled and you'll only end up buying 390 shares in total.

```
C#
LimitOrder(
    _symbol, quantity, limitPrice,
    orderProperties: new InteractiveBrokersOrderProperties
    {
        FaMethod = "Equal"
    }
);
```

Net Liquidation Value

This group allocation method distributes shares based on the net liquidation value of each account. The system calculates ratios based on the net liquidation value in each account and allocates shares based on these ratios. When you use this method, you need to specify an order quantity.

For example, say your account group includes three accounts, A, B and C with Net Liquidation values of 25,000, 50,000 and $100,000, respectively. In this case, the system calculates a ratio of 1:2:4. If you place an order for 700 shares of a stock, it allocates 100 shares to Client A, 200 shares to Client B, and 400 shares to Client C.

```
C#
LimitOrder(_symbol, quantity, limitPrice,
    orderProperties: new InteractiveBrokersOrderProperties
    {
        FaMethod = "NetLiq"
    }
);
```

Available Equity

This group allocation method distributes shares based on the amount of available equity in each account. The system calculates ratios based on the available equity in each account and allocates shares based on these ratios. When you use this method, you need to specify an order quantity.

For example, say your account group includes three accounts, A, B and C with available equity of 25,000, 50,000 and $100,000, respectively. In this case, the system calculates a ratio of 1:2:4. If you place an order for 700 shares of a stock, it allocates 100 shares to Client A, 200 shares to Client B, and 400 shares to Client C.

```
C#
LimitOrder(_symbol, quantity, limitPrice,
    orderProperties: new InteractiveBrokersOrderProperties
    {
        FaMethod = "AvailableEquity"
    }
);
```

Subscription Requirements

To use FA group orders through our Interactive Brokers integration, you need to connect as a member of a Trading Firm and Institution organization. If you aren't currently on either of these tiers, upgrade your organization.

Live Trading

Reconciliation

Introduction

Algorithms usually perform differently between backtesting and live trading over the same time period. Backtests are simulations where we model reality as close as possible, but the modeling isn't always perfect. To measure the performance differences, we run an out-of-sample (OSS) backtest in parallel to all of your live trading deployments. The live results page displays the live equity curve and the OOS backtest equity curve of your algorithms.

If your algorithm is perfectly reconciled, it has an exact overlap between its live and OOS backtest equity curves. Deviations mean that the performance of your algorithm has differed between the two execution modes. Several factors can contribute to the deviations.

Differences From Data

The data that your algorithm uses can cause differences between backtesting and live trading performance.

Data Providers

QuantConnect provides high-quality historical data for all supported asset classes . However, our live data provider lacks data feeds for equity options, index options, and futures options. If your algorithm trades these asset classes, you must rely on brokerage or third-party data feeds.

Look-Ahead Bias

The Time Frontier minimizes the risk of look-ahead bias in backtests, but it does not completely eliminate the risk of look-ahead bias. For instance, if you use a custom dataset that contains look-ahead bias, your algorithm's live and backtest equity curves

may deviate. To avoid look-ahead bias with custom datasets, set a `Period` on your custom data points so that your algorithm receives the data points after the `Time + Period`.

Discrete Time Steps

In backtests, we inject data into your algorithm at predictable times, according to the data resolution. In live trading, we inject data into your algorithm when new data is available. Therefore, if your algorithm has a condition with a specific time (i.e. time is 9:30:15), the condition may work in backtests but it will always fail in live trading since live data has microsecond precision. To avoid issues, either use a time range in your condition (i.e. 9:30:10 < time < 9:30:20), use a rounded time, or use a Scheduled Event.

Custom Data Emission Times

Custom data is often timestamped to midnight, but the data point may not be available in reality until several days after that point. If your custom dataset is prone to this delay, your backtest may not fetch the same data at the same time or frequency that your live trading algorithm receives the data, leading to deviations between backtesting and live trading. To avoid issues, ensure the timestamps of your custom dataset are the times when the data points would be available in reality.

In backtesting, LEAN and custom data are perfectly synchonized. In live trading, daily and hourly data from a custom data source are not because of the frequency that LEAN checks the data source depends on the `resolution` argument. The following table shows the polling frequency of each resolution:

Resolution	**Update Frequency**
Daily	Every 30 minutes
Hour	Every 30 minutes
Minute	Every minute
Second	Every second
Tick	Constantly checks for new data

Split Adjustment of Indicators

Backtests use adjusted price data by default. Therefore, if you don't change the data normalization mode , the indicators in your backtests are updated with adjusted price data. In contrast, if a split or dividend occurs in live trading, your indicators will temporarily contain price data from before the corporate event and price data from after the corporate event. If this occurs, your indicators will produce different signals in your backtests compared to your live trading deployment. To avoid issues, reset and warm up your indicators when your algorithm receives a corporate event.

Tick Slice Sizes

In backtesting, we collect ticks into slices that span 1 millisecond before injecting them into your algorithm. In live trading, we collect ticks into slices that span up to 70 milliseconds before injecting them into your algorithm. This difference in slice sizes can cause deviations between your algorithm's live and OOS backtest equity curves. To avoid issues, ensure your strategy logic is compatible with both slice sizes.

Opening and Closing Auctions

The opening and closing price of the day is set by very specific opening and closing auction ticks. When a stock like Apple is listed, it’s listed on Nasdaq. The open auction tick on Nasdaq is the price that’s used as the official open of the day. NYSE,

BATS, and other exchanges also have opening auctions, but the only official opening price for Apple is the opening auction on the exchange where it was listed.

We set the opening and closing prices of the first and last bars of the day to the official auction prices. This process is used for second, minute, hour, and daily bars for the 9:30 AM and 4:00 PM Eastern Time (ET) prices. In contrast, other platforms might not be using the correct opening and closing prices.

The official auction prices are usually emitted 2-30 seconds after the market open and close. We do our best to use the official opening and closing prices in the bars we build, but the delay can be so large that there isn't enough time to update the opening and closing price of the bar before it's injected into your algorithms. For example, if you subscribe to second resolution data, we wait until the end of the second for the opening price but most second resolution data won't get the official opening price. If you subscribe to minute resolution data, we wait until the end of the minute for the opening auction price. Most of the time, you'll get the actual opening auction price with minute resolution data, but there are always exceptions. Nasdaq and NYSE can have delays in publishing the opening auction price, but we don't have control over those issues and we have to emit the data on time so that you get the bar you are expecting.

Data Updates

Data issues are incorrect or missing values in a dataset. These issues are generally a result of human error or from mistakes in the data collection process. Data issues can be reported by any QuantConnect member. When data issues are reported and verified, our Data Team works to quickly resolve them. Thanks to the communal efforts of the QuantConnect members, the QuantConnect data is reviewed and fixed by over 250,000 people, giving you a very high-quality source of data.

Data updates change backtest results and, consequently, cause differences between backtesting running with fixed data and live trading. For more information on common data issues, please refer to the data issues documentation .

Differences From Modeling

The modeling that your algorithm uses can cause differences between backtesting and live trading performance.

Reality Modeling Error

We provide brokerage models to model fees, slippage, and order fills in backtests. However, these model predictions may not always match the fees that your live algorithm incurs, leading to deviations between backtesting and live trading. You can adjust the reality models that your algorithm uses to more accurately reflect the specific assets that you're trading. For more information about reality models, see Reality Modeling .

Market Impact

We don't currently model market impact. So, if you are trading large orders, your fill prices can be better during backtesting than live trading, causing deviations between backtesting and live trading. To avoid issues, implement a custom fill model in your backtests that incorporates market impact.

Fills

In backtests, orders fill immediately. In live trading, they are sent to your brokerage and take about half a second to execute. If you fill an order in a backtest with stale data, deviations between backtesting and live trading can occur because the order is filled at a price that is likely different from the real market price. Stale order fills commonly occur in backtests when you create a Scheduled Event with an incompatible data resolution. For instance, if you subscribe to hourly data, place a Scheduled Event for 11:15 AM, and fill an order during the Scheduled Event, the order will fill at a stale price because the data between 11:00 AM and 11:15 AM is missing. To avoid stale fills, only place orders when your algorithm receives price data.

In live trading, your brokerage provides the fill price of your orders. Since the backtesting brokerage models do not know the

price at which live orders are filled, the fill price of backtest orders is based on the best price available in the current backtesting data. Similarly, limit orders can fill at different prices between backtesting and live trading. In backtesting, limit orders fill as soon as the limit price is hit. In live trading, your brokerage may fill the same limit order at a different price or fail to fill the order, depending on the position of your order in their order book.

Borrowing Costs

By default, LEAN doesn't simulate the cost of borrowing shorts in backtests, but you can enable them by setting a margin interest model.

Differences From Brokerage

The brokerage that your algorithm uses can cause differences between backtesting and live trading performance.

Portfolio Allocations on Small Accounts

If you trade a small portfolio, it's difficult to achieve accurate portfolio allocations because shares are usually sold in whole numbers. For instance, you likely can't allocate exactly 10% of your portfolio to a security. You can use fractional shares to achieve accurate portfolio allocations, but not all brokerages support fractional shares. To get the closest results when backtesting and live trading over the same period, ensure both algorithms have the same starting cash balance.

Different Backtest Parameters

If you don't start your backtest and live deployment on the same date with the same holdings, deviations can occur between backtesting and live trading. To avoid issues, ensure your backtest parameters are the same as your live deployment.

Non-deterministic State From Algorithm Restarts

If you stop and redeploy your live trading algorithm, it needs to restart in a stateful way or else deviations can occur between backtesting and live trading. To avoid issues, redeploy your algorithm in a stateful way using the `SetWarmUp` and `History` methods. Furthermore, use the Object Store to save state information between your live trading deployments.

Existing Portfolio Securities

If you deploy your algorithm to live trading with a brokerage account that has existing holdings, your live trading equity curve reflects your existing positions, but the backtesting curve won't. Therefore, if you have existing positions in your brokerage account when you deploy your algorithm to live trading, deviations will occur between backtesting and live trading. To avoid issues, deploy your algorithm to live trading using a separate brokerage account or subaccount that does not have existing positions.

Brokerage Limitations

We provide brokerage models that support specific order types and model your buying power. In backtesting, we simulate your orders with the brokerage model you select. In live trading, we send your orders to your brokerage for execution. If the brokerage model that you use in backtesting is not the same brokerage that you use in live trading, deviations may occur between backtesting and live trading. The deviations can occur if your live brokerage doesn't support the order types that you use or if the backtesting brokerage models your buying power with a different methodology than the real brokerage. To avoid brokerage model issues, set the brokerage model in your backtest to the same brokerage that you use in live trading.

Differences From Third-Party Indicators

The values of our indicators can sometimes produce different results than the indicators on other platforms. There can be several reasons for these discrepancies.

Price Differences

If you find differences in indicator values, compare the prices that are fed into the indicator on each platform. If the input data is slightly different, the indicator values will be different. To test if it's a difference in price data, feed in the data from the third-party platform into our indicators. We validate the indicators against third-party sources and when the values are the same, the indicator values are similar too.

Timestamp Differences

We timestamp our data to the time when the period ends. Many other platforms timestamp to the beginning of the candle.

Implementation Differences

Some indicators can have slightly different default arguments for their implementation. A common difference across platforms is to use a different `MovingAverageType` for the indicators. To view the default arguments for all our indicators, see Supported Indicators . To view the full implementation of our indicators, see the LEAN GitHub repository .

Warm Up Period Differences

Some platforms use all of the historical data to warm up their indicators while LEAN indicators have a fixed number of bars they need to warm up. As a result, indicators with long memory like the Exponential Moving Average can have slightly different values across platforms.

Risk Free Interest Rate Differences

Some indicators, like Sharpe ratios and Sortino ratios , are a function of the risk free interest rate . LEAN provides several different ways to define the interest rate, which can lead to different indicator values compared to other platforms.

Differences From Real-time Scheduled Events

In live trading, Scheduled Events execute in a parallel thread based on a real-time clock. If you set a Scheduled Event to fire at 10:00 AM, it executes at exactly 10:00 AM. In backtesting, Scheduled Events are part of the main algorithm manager loop, so they may not execute exactly when you set them. For example, if your algorithm subscribes to minute resolution US Equity data with regular trading hours and you set a Scheduled Event to occur at 2:00 AM, your Scheduled Event will execute at 9:31 AM when the next bar is fed into your algorithm.

The difference between live trading and backtesting is important to note because it can affect your algorithm's behavior. There are two common scenarios to consider.

Execution Timing and Backtest Timeouts

Take the following scenario:

- You set Scheduled Events for 2:00 AM, 3:00 AM, and 4:00 AM each day.
- Each Scheduled Event takes eight minutes to execute.
- Your algorithm only subscribes to US Equity securities without extended market hours (9:30 AM - 4:00 PM).

In this scenario, the Scheduled Events each fire at the correct time and execute without error in live trading. In backtesting, all of the Scheduled Events execute at 9:31 AM when your algorithm receives the first bar of the trading day. Since all the Scheduled Events take eight minutes to execute, the algorithm tries to execute all the Scheduled Events but reaches the 10-minute timeout and the backtest stops execution.

Live Data Delays

In backtests, your algorithm receives data at perfect timing. If you request minute resolution data, your algorithm receives the

bars at the top of each minute. In live trading, bars have a slight delay, so you may receive them milliseconds after the top of each minute. Take the following scenario:

- You subscribe to minute resolution data
- You set a Scheduled Event for 10:00 AM
- The Scheduled Event checks the current asset price

In live trading, the Scheduled Event executes at exactly 10:00 AM but your algorithm may receive the 9:59-10:00 AM bar at 10:00:00.01 AM. Therefore, when you check the price in the Scheduled Event, the price from the 9:58-9:59 AM bar is the latest price. In backtesting, the Scheduled Event gets the price from the 9:59-10:00 AM bar since your algorithm receives the bar at perfect timing.

Live Trading

Notifications

Introduction

Set up some live trading notifications so that you are notified of market events and your algorithm's performance. QuantConnect Cloud supports email, SMS, webhooks, and Telegram notifications for live trading algorithms. If you run local algorithm with LEAN or the LEAN CLI, the notification services aren't available. To view the number of notification you can send for free in QuantConnect Cloud, see the Live Trading Notification Quotas . Notifications are different from commands because notifications enable you to send information out of your algorithm while commands enable you to manually call methods in your algorithm as it runs.

Email

Email notifications can include up to 10KB of text content in the message body. These notifications can be slow since they go through your email provider. If you don't receive an email notification that you're expecting, check your junk folders.

To send email notifications in your code files, run:

```C#
var message = "2023-07-13T12:20:00.2033226Z,EURUSD,1.11853,89187,Market,Filled,99758.33511";
Notify.Email(emailAddress: "quant@office.com", subject: "Last Trade", message: message);
```

You can include an attachment and set its name.

```C#
var attachment = "2023-07-13T12:20:00.2033226Z,EURUSD,1.11853,89187,Market,Filled,99758.33511";
var headers = new Dictionary<string, string> { { "filename", "trades.csv" } };
Notify.Email(address: "quant@office.com", subject: "Last Trade", message: "See attachment", data: attachment,
headers: headers);
```

If you don't provide headers, the attachment name is attachment.txt .

FTP

FTP notifications send files to your FTP server. You can use password and SSH authentication.

To send a file to your FTP server in your code files, run:

```C#
var fileContent = "2023-07-13T12:20:00.2033226Z,EURUSD,1.11853,89187,Market,Filled,99758.33511";
Notify.Ftp(hostname: "myftpserver.com", username: "admin", password: "12345", filePath: "trades.csv", fileContent:
fileContent, port: 21);
```

To send a file to your SFTP server in your code files, run:

```C#
var fileContent = "2023-07-13T12:20:00.2033226Z,EURUSD,1.11853,89187,Market,Filled,99758.33511";
Notify.Sftp(hostname: "mysftpserver.com", username: "admin", password: "12345", filePath: "trades.csv", fileContent:
fileContent, port: 21);
```

You can use password authentication over SFTP using SSH keys.

```C#
var fileContent = "2023-07-13T12:20:00.2033226Z,EURUSD,1.11853,89187,Market,Filled,99758.33511";
Notify.Sftp(hostname: "mysftpserver.com", username: "admin", privateKey: "12345", privateKeyPassphrase: "67890",
filePath: "trades.csv", fileContent: fileContent, port: 21);
```

The `filePath` parameter is path to file on the FTP server. The `port` parameter is optional.

SMS

SMS notifications are the only type of notification that you don't need an internet connection to receive. They can include up to 1,600 characters of text content in the message body.

To send SMS notifications in your code files, run:

```C#
var message = "2023-07-13T12:20:00.2033226Z,EURUSD,1.11853,89187,Market,Filled,99758.33511";
Notify.Sms(phoneNumber: "+100503016366", message: message);
```

Telegram

Telegram notifications are automated messages to a Telegram group.

To send Telegram notifications in your code files, run:

```C#
var message = "2023-07-13T12:20:00.2033226Z,EURUSD,1.11853,89187,Market,Filled,99758.33511";
Notify.Telegram(id: "-503016366", message: message, token: "1234");
```

The `id` argument is your Telegram group Id. Your group Id is in the URL when you open your group chat in the Telegram web interface. For example, the group Id of web.telegram.org/z/#-503016366 is -503016366.

The `token` optional argument is a token of a bot you must add to your Telegram group. To create a bot, chat with @BotFather and follow its instructions. If you want to use our bot, the username is @quantconnect_notifications_bot.

To add emoticons to your `message` , convert the emoticon character to UTF-32 with the Unicode Converter on the Branah website. For example, the rocket emoji is `0001f680` in UTF-32 format, so use `\U001f680` in your message.

Webhooks

Webhook notifications are an HTTP-POST request to a URL you provide. The request is sent with a timeout of 300s. You can process these notifications on your web server however you want. For instance, you can inject the content of the notifications into your server's database or use it to create other notifications on your own server.

To send webhook notifications in your code files, run:

```C#
var data = "";
var headers = new Dictionary<string, string> { { "Authorization", "Basic XXXXXX" } };
Notify.Web(address: "http://api.myserver.com", data: data, headers: headers);
```

The `data` argument can be any serializable type .

To send a webhook to a Discord server, generate a webhook URL (for instructions, see Intro to Webhooks in the Discord Help

Center) and serialize the data in a dictionary with a `content` key.

```C#
var discordURL = "https://discord.com/api/webhooks/.../...";
Notify.Web(discordURL, JsonSerializer.Serialize(new { content = "Hello world!" }));
```

Receive Notifications

To receive external notification in your algorithm, see Commands .

Terms of Use

The notification system can't be used for data distribution.

Examples

The following examples demonstrate common practices for sending notifications in live algorithms.

Example 1: Pre-Open Order Error

The following algorithm simulate extended market hour trading on Tradier Brokerage . We place a market-on-open order at 7:30am and expect the order cannot submit to the broker since it is not supported, resulting into an error.

```C#
public class OrderErrorsAlgorithm : QCAlgorithm
{
    private Symbol _spy;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 9, 5);
        // Simulate Tradier brokerage, which does not support the market-on-open orders.
        SetBrokerageModel(BrokerageName.TradierBrokerage, AccountType.Cash);

        // Request extended market hour SPY data for trading.
        _spy = AddEquity("SPY", extendedMarketHours: true).Symbol;

        // Set a scheduled event to trade 2 hours pre-open to place the market-on-open-order.
        Schedule.On(
            DateRules.EveryDay(_spy),
            TimeRules.BeforeMarketOpen(_spy, 120),
            OpenPosition
        );
    }

    private void OpenPosition()
    {
        // Buy on market open will result in an error since Tradier does not support it.
        MarketOnOpenOrder(_spy, 10);
    }

    // It will be triggered on live trading.
    public override void OnBrokerageMessage(BrokerageMessageEvent message)
    {
        if (message.Type == BrokerageMessageType.Error)
        {
            Log($"{Time}: {message.Type}: Message: {message.Message}");
        }
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm
LiveFeaturesAlgorithm.cs C#

Live Trading

Commands

Introduction

Commands enable you to manually call methods in your live algorithm as it runs. You can command live projects to update your algorithm state, place orders, or run any other logic. Commands are different from notifications because notifications enable you to send information *out* of your algorithm while commands enable you to send information *into* your algorithm.

Basic Usage

To receive and handle generic commands in your algorithm, define the `OnCommand` method. This method receives the data you want to inject into your algorithm and returns a boolean that represents if the command was successful.

```C#
public class BasicCommandAlgorithm : QCAlgorithm
{

    public override void Initialize()
    {
        SetBenchmark(x => 1); // So the algorithm needs no asset data.
    }

    public override bool? OnCommand(dynamic data)
    {
        Log($"Got command at {Time} with data: {data}");
        return true;
    }
}
```

To invoke the `OnCommand` event handler, send a payload (for example, `{"ticker": "AAPL", "quantity": 1}`) through the QuantConnect REST API or LEAN CLI . The keys of the payload become the members of the `data` object.

Encapsulate Event Handlers

To encapsulate the command handler logic, define isolated classes. Command classes should extend the `Command` class. Extensions of the `Command` class must implement a `Run` method, which receives the payload you send and returns a boolean that represents if the command was successful. To add the command to your algorithm, call the `AddCommand` method.

```C#
public class EncapsulatedCommandAlgorithm : QCAlgorithm
{

    public override void Initialize()
    {
        AddCommand<MyCommand>();
        SetBenchmark(x => 1); // So the algorithm doesn't need asset data.
    }

    public void DoSomething()
    {
        Log("Something was done!");
    }
}

public class MyCommand : Command
{
    public string Ticker { get; set; }
    public int Quantity { get; set; }
    public Dictionary<string, string> Parameters { get; set; }
    public override bool? Run(IAlgorithm algorithm)
    {
        ((QCAlgorithm)algorithm).Log($"ticker: {Ticker}; quantity: {Quantity}; parameters: {Parameters}");
        ((EncapsulatedCommandAlgorithm)algorithm).DoSomething();
        return true;
    }
}
```

To invoke the `Run` method of the class you define, send a payload with a `$type` key (for example, `{"ticker": "AAPL", "quantity": 1, "$type": "MyCommand"}`). The value of the `$type` key must be the `Command` class you define. The other keys of the payload set the members of the class. If you don't provide a `$type` key, LEAN runs the `OnCommand` method instead of the `Run` method in your custom `Command` class.

Send Command Link

You can create the command in your algorithm and then send yourself an email notification with a link inside it. When you click the link, it then executes the event handler in your algorithm. To create the link, call the `Link` method. The following algorithm demonstrates how send command requests for the `on_command` event handler:

```C#
public class BasicLinkedCommandAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetBenchmark(x => 1); // So the algorithm needs no asset data.
        var link = Link(new {Ticker = "AAPL", Quantity = 1});
        Notify.Email("email@address.com", "Run Command?", $"Click here to run: {link}");
    }

    public override bool? OnCommand(dynamic data)
    {
        Log($"Got command at {Time} with data: {data}");
        return true;
    }
}
```

The following algorithm demonstrates how to send command requests for an event handler you encapsulate in a class:

```C#
public class EncapsulatedAndLinkedCommandAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        AddCommand<MyCommand>();
        SetBenchmark(x => 1); // So the algorithm doesn't need asset data.
        var potentialCommand = new MyCommand
        {
            Ticker = "AAPL",
            Quantity = 1,
            Parameters = new() { {"tag", "Signal X"} }
        };
        var link = Link(potentialCommand);
        Notify.Email("email@address.com", "Run Command?", $"Click here to run: {link}");
    }

}

public class MyCommand : Command
{
    public string Ticker { get; set; }
    public int Quantity { get; set; }
    public Dictionary<string, string> Parameters { get; set; }
    public override bool? Run(IAlgorithm algorithm)
    {
        ((QCAlgorithm)algorithm).Log($"ticker: {Ticker}; quantity: {Quantity}; parameters: {Parameters}");
        return true;
    }
}
```

Broadcast Command

You can create the command in your algorithm and then broadcast it to all live algorithms in your organization, except the source algorithm and commands of unregisted type name. The following algorithm demonstrates how broadcast a command to the `OnCommand` event handler in all your live algorithms:

```C#
public class BasicBroadcastCommandAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetBenchmark(x => 1); // So the algorithm needs no asset data.
        var broadcastResult = BroadcastCommand(new {Ticker = "AAPL", Quantity = 1});
    }

    public override bool? OnCommand(dynamic data)
    {
        Log($"Got command at {Time} with data: {data}");
        return true;
    }
}
```

You can control the command data by encapsulating the event handler in a class. The following algorithm demonstrates how to broadcast a command for an event handler you encapsulate in a class:

```C#
public class EncapsulatedAndBroadcastCommandAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        AddCommand<MyCommand>();
        SetBenchmark(x => 1); // So the algorithm doesn't need asset data.
        var potentialCommand = new MyCommand
        {
            Ticker = "AAPL",
            Quantity = 1,
            Parameters = new() { {"tag", "Signal X"} }
        };
        var broadcastResult = BroadcastCommand(potentialCommand);
    }
}

public class MyCommand : Command
{
    public string Ticker { get; set; }
    public int Quantity { get; set; }
    public Dictionary<string, string> Parameters { get; set; }
    public override bool? Run(IAlgorithm algorithm)
    {
        ((QCAlgorithm)algorithm).Log($"ticker: {Ticker}; quantity: {Quantity}; parameters: {Parameters}");
        return true;
    }
}
```

Send Commands by API

To send a command to your algorithm, send a payload through the QuantConnect REST API or LEAN CLI . LEAN converts the first character in each key of the payload to uppercase before injecting the data into your algorithm. The following script demonstrates how to send a command with the REST API. In this case, it calls the `OnCommand` method.

To get your user Id and API token, see Request API Token .

Broadcast Commands by API

To broadcast a command to your live algorithms, send a payload through the QuantConnect REST API or LEAN CLI . LEAN converts the first character in each key of the payload to uppercase before injecting the data into your algorithm. The following script demonstrates how to broadcast a command to all projects in your organization with the REST API.

To get your user Id, API token, and organization Id, see Request API Token and Get Organization Id .

Examples

The following examples demonstrate common practices for implementing live commands.

Example 1: Grey Box

The following algorithm trades a simple EMA cross strategy on SPY. However, instead of direct ordering by the algorithm as a "black box", it sends an email to ask for clicking a confirmation link before ordering, making each order decision an informed manual decision. The algorithm is called a "grey box".

```C#
public class LiveCommandAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private ExponentialMovingAverage _ema;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 12);
        SetEndDate(2024, 10, 1);
        SetCash(1000000);

        // Request SPY data to trade.
        _spy = AddEquity("SPY").Symbol;
        // Create an EMA indicator to generate trade signals.
        _ema = EMA(_spy, 20, Resolution.Daily);
        // Warm-up indicator for immediate readiness.
        WarmUpIndicator(_spy, _ema, Resolution.Daily);
    }

    public override void OnData(Slice slice)
    {
        if (slice.Bars.TryGetValue(_spy, out var bar) && LiveMode)
        {
            // Trend-following strategy using price and EMA.
            // If the price is above EMA, SPY is in an uptrend, and we buy it.
            // We sent a link to our email address and await confirmation.
            string link;
            if (bar.Close > _ema && !Portfolio[_spy].IsLong)
            {
                link = Link(new {Ticker = "SPY", Size = 1});
                Notify.Email("email@address.com", "Trade Confirmation Needed", $"Click here to run: {link}");
            }
            else if (bar.Close < _ema && !Portfolio[_spy].IsShort)
            {
                link = Link(new {Ticker = "SPY", Size = -1});
                Notify.Email("email@address.com", "Trade Confirmation Needed", $"Click here to run: {link}");
            }
        }
    }

    public override bool? OnCommand(dynamic data)
    {
        // If we click the email link to confirm the trade, the algorithm will place the order.
        SetHoldings(data.Ticker, data.Size);
        return true;
    }
}
```

Example 2: Parent-Child Communication

Parent algorithm

The parent algorithm is connected to a brokerage. It receives commands from other algorithms and places market orders based on those commands.

```C#
public class ParentAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);

        // Remove benchmark so doesn't need asset data.
        SetBenchmark(x => 1);

        // Seed each asset with it's price to trade right away.
        Settings.SeedInitialPrices = true;
    }

    public override bool? OnCommand(dynamic payload)
    {
        Log($"{Time} - Execute on_command(): {payload}");
        var equity = AddEquity(payload.ticker);

        if (!string.IsNullOrWhiteSpace(payload.email))
        {
            Notify.Email(address: payload.email, subject: $"Order Executed from {payload.source}", message: $"
{equity.symbol}, {payload.quantity}, {payload.source}");
        }

        MarketOrder(equity.symbol, payload.quantity, asynchronous: true, tag: payload.source);
        return true;
    }
}
```

Child algorithm

The child algorithm is connected to a paper trading account. It contains the trading logic and sends commands to the parent algorithm for execution.

```C#
public class ChildAlgorithm : QCAlgorithm
{
    // Set the initials to identify the sender and the cash this algorithm will use.
    private const string _initials = "C1";

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetCash(100000);
    }

    public override void OnOrderEvent(OrderEvent orderEvent)
    {
        // When an order is filled, we send a command to the parent algorithm with the order details.
        if (orderEvent.Status == OrderStatus.Filled)
        {
            Log($"{Time} - Sending command");
            var response = Send(orderEvent.Symbol.Value, orderEvent.FillQuantity, _initials);
            Log($"{Time} - Command response {response}");
        }
    }

    private string Send(string ticker, decimal quantity, string name)
    {
        if (!LiveMode) return string.Empty;

        // Set the project ID with the ID of your parent algorithm.
        ProjectId = 30123456;
        // Generate the command link with the desired data payload, and execute with download_data.
        var link = Link(new {ticker, quantity, name});
        return link.DownloadData();
    }
}
```

Other Examples

For more examples, see the following algorithms:

Demonstration Algorithm
CallbackCommandRegressionAlgorithm.cs C#

Live Trading

Arbitrage

Introduction

Arbitrage is the practice of taking advantage of a price difference between two or more markets. QuantConnect supports arbitrage strategies by allowing you to run multiple live algorithms deployments simultaneously and communicate between them using commands .

Basic Implementation

You can broadcast a command from and execute the event handler in another project. Add the project ID to the command data to identify the source of the command:

```C#
public class BasicBroadcastCommandAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        SetBenchmark(x => 0);           // So the algorithm needs no asset data.
        var broadcastResult = BroadcastCommand(new { Sender = ProjectId, Ticker = "AAPL", Quantity = 1 });
    }

    public override bool? OnCommand(dynamic data)
    {
        // Receive the command from the other algorithm
        Log($"Got command at {Time} with data: {data}");
        return true;
    }
}
```

Examples

The following examples demonstrate common practices for implementing arbitrage algorithms.

Example 1: Long-short Bitcoin in two exchanges

The following algorithms trade an arbitrage strategy with BTCUSDT.

Primary algorithm

The primary algorithm trades a simple EMA cross strategy on BTCUSDT in Kraken. It broadcasts a command with the order filled quantity to all live deployments, and logs the command data it receives from the secondary algorithm.

C#

```
public class CryptoArbitrageA : QCAlgorithm
{
    private Symbol _btc;
    private ExponentialMovingAverage _ema;
    // Set a threshold value for the EMA cross.
    private decimal _threshold = 0.005m;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetAccountCurrency("USDT", 100000);

        // Set the brokerage model to trade on Kraken.
        SetBrokerageModel(BrokerageName.Kraken, AccountType.Margin);
        // Request BTCUSDT data to trade.
        _btc = AddCrypto("BTCUSDT").Symbol;
        // Create an EMA indicator to generate trade signals.
        _ema = EMA(_btc, 22);
        // Warm-up indicator for immediate readiness.
        WarmUpIndicator(_btc, _ema);
        // Trade when the EMA is updated.
        _ema.Updated += Trade;
    }

    private void Trade(object _, IndicatorDataPoint current)
    {
        var ema = current.Value;
        var holdings = Portfolio[_btc];
        var price = holdings.Price;

        // Trend-following strategy using price and EMA.
        // If the price is above EMA, BTC is in an uptrend, and we buy it.
        if (!holdings.IsLong && price > ema * (1m+_threshold))
        {
            SetHoldings(_btc, 1);
        }
        else if (!holdings.IsShort && price < ema * (1m-_threshold))
        {
            SetHoldings(_btc, -1);
        }
    }

    public override bool? OnCommand(dynamic data)
    {
        // The algorithm will log the command data from the secondary algorithm.
        Log($"Got command at {Time} with data: {data}");
        return true;
    }

    public override void OnOrderEvent(OrderEvent orderEvent)
    {
        // If an order is filled, we call the BroadcastCommand method to send a command to all live deployments.
        // You can add the project ID to the command data to identigy the sender.
        if (!LiveMode || orderEvent.Status != OrderStatus.Filled) return;
        var broadcastResult = BroadcastCommand(new { Sender = ProjectId, Ticker = $"{orderEvent.Symbol.Value}",
Portfolio[orderEvent.Symbol].Quantity });
    }

}
```

Secondary algorithm

The secondary algorithm trades the opposite direction of the primary in Bybit using the command it receives from the primary algorithm. It broadcasts a command with the order filled quantity to all live deployments.

C#

```
public class CryptoArbitrageB : QCAlgorithm
{
    private Symbol _btc;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 1);
        SetEndDate(2024, 12, 31);
        SetAccountCurrency("USDT", 100000);

        // Set the brokerage model to trade on Bybit.
        SetBrokerageModel(BrokerageName.Bybit, AccountType.Margin);
        // Request BTCUSDT data to trade.
        _btc = AddCrypto("BTCUSDT").Symbol;
    }

    public override bool? OnCommand(dynamic data)
    {
        // The algorithm will trade the opposite direction of the primary algorithm.
        Log($"Got command at {Time} with data: {data}");
        var quantity = -(Portfolio[_btc].Quantity + (decimal)data.Quantity);
        if (quantity != 0)
            MarketOrder(_btc, quantity, tag:data.ToString());
        return true;
    }

    public override void OnOrderEvent(OrderEvent orderEvent)
    {
        // If an order is filled, we call the BroadcastCommand method to send a command to all live deployments.
        // You can add the project ID to the command data to identify the sender.
        if (!LiveMode || orderEvent.Status != OrderStatus.Filled) return;
        var broadcastResult = BroadcastCommand(new { Sender = ProjectId, Ticker = $"{orderEvent.Symbol.Value}",
Portfolio[orderEvent.Symbol].Quantity });
    }

}
```

Live Trading

Charting and Logging

Introduction

The live results page shows your algorithm's live trading performance. You can add custom charts and logs to the results page.

Charting

Charts are sampled every one and ten minutes. If you create 1-minute resolution custom charts, the IDE charting will downgrade the granularity and display the 10-minutes sampling after a certain amount of samples. There are not quotas in live trading.

Logging

Algorithms can record string messages ('log statements') to a file for analysis after a backtest is complete, or as a live algorithm is running. These records can assist in debugging logical flow errors in the project code. Consider adding them in the code block of an `if` statement to signify an error has been caught.

It's good practice to add logging statements to live algorithms so you can understand its behavior and keep records to compare against backtest results. If you don't add logging statements to a live algorithm and the algorithm doesn't trade as you expect, it's difficult to evaluate the underlying problem.

If you run algorithms on QuantConnect, you must stay within the log quota . To only log when your algorithm is live, use the `LiveMode` property.

C#

```
if (LiveMode)
{
    Log("My log message");
}
```

Runtime Statistics

Runtime statistics show the performace of your algorithm at a single moment in time.

The following table describes the default runtime statistics:

Statistic	Description
Equity	The total portfolio value if all of the holdings were sold at current market rates.
Fees	The total quantity of fees paid for all the transactions.
Holdings	The absolute sum of the items in the portfolio.
Net Profit	The dollar-value return across the entire trading period.
PSR	The probability that the estimated Sharpe ratio of an algorithm is greater than a benchmark (1).
Return	The rate of return across the entire trading period.
Unrealized	The amount of profit a portfolio would capture if it liquidated all open positions and paid the fees for transacting and crossing the spread.
Volume	The total value of assets traded for all of an algorithm's transactions.

For more information about runtime statistics, see Runtime Statistics .

Examples

The following examples demonstrate some common practices for logging in live algorithm.

Example 1: On Events And Trading Logic

The following example trades a simple EMA cross trend-following strategy on SPY. We log or debug messages in event handlers and trading logic changes to get information on any event flow.

C#

```
public class LoggingAlgorithm : QCAlgorithm
{
    private Symbol _spy;
    private ExponentialMovingAverage _ema;

    public override void Initialize()
    {
        SetStartDate(2024, 9, 12);
        SetEndDate(2024, 10, 1);
        SetCash(1000000);

        // Request SPY data to trade.
        _spy = AddEquity("SPY").Symbol;
        // Create an EMA indicator to generate trade signals.
        _ema = EMA(_spy, 20, Resolution.Daily);

        // Warm up the algorithm for indicator readiness.
        SetWarmUp(20, Resolution.Daily);
    }

    public override void OnWarmupFinished()
    {
        // Signals on warm-up finished.
        Log("Warm up finished");
    }

    public override void OnData(Slice slice)
    {
        if (!IsWarmingUp && slice.Bars.TryGetValue(_spy, out var bar))
        {
            // Trend-following strategy using price and EMA.
            // If the price is above EMA, SPY is in an uptrend, and we buy it.
            if (bar.Close > _ema && Portfolio[_spy].IsLong)
            {
                Debug("Trend changes to upwards");
                SetHoldings(_spy, 1m);
            }
            else if (bar.Close < _ema && Portfolio[_spy].IsShort)
            {
                Debug("Trend changes to downwards");
                SetHoldings(_spy, -1m);
            }
        }
    }

    public override void OnOrderEvent(OrderEvent orderEvent)
    {
        if (orderEvent.Status == OrderStatus.Filled)
        {
            // Log the order details if being filled.
            Log($"Order filled - Quantity: {orderEvent.Quantity}");
        }
    }
}
```

Live Trading

Signal Exports

QuantConnect Signal Exports allows you to collect distributed, crowd-sourced, algorithm algorithm's trading and portfolio positions to your fund or a third-party service..

Key Concepts

Getting started with Signal Exports

Collective2

Equities, Equity Options, Forex, Futures, Index Options

Numerai

Equities

vBase

See Also

Datasets

Signal Exports

Key Concepts

Introduction

QuantConnect Signal Exports allows you to transmit the state of an algorithm's trading and portfolio positions to your fund or a third-party service. QuantConnect implements two signal services out of the box: Collective2 and Numerai . To send portfolio targets to your fund, you need to create a custom Signal Export implementation.

Add Providers

To export signals to a signal provider, during initialization , add a signal export provider.

```
C#
SignalExport.AddSignalExportProvider(new CustomSignalExport());
```

You can add multiple signal export providers to a single algorithm.

Send Portfolio Targets

The signal export manager automatically sends signals when your portfolio holdings change to your fund or a third-party service. By default, it waits five seconds after an order is filled to aggregate all the state changes into a single post. Set the `AutomaticExportTimeSpan` property to change the waiting time.

```
C#
SignalExport.AutomaticExportTimeSpan = TimeSpan.FromSeconds(1);
```

To send targets that aren't based on your portfolio holdings, see Manual Exports .

To disable automatic exports, set the `AutomaticExportTimeSpan` property to `null` .

```
C#
SignalExport.AutomaticExportTimeSpan = null;
```

Custom Signal Exports

Signal export providers must implement the `ISignalExportTarget` interface. Extensions of the `ISignalExportTarget` interface must define the `Send` and the `Dispose` methods.

The `Send` receives a `SignalExportTargetParameters` object with a list of `PortfolioTarget` objects you can use to create the API request payload.

The `Dispose` method is called when the algorithm is disposed. You can use this method to release unmanaged resources such as HTTP Clients.

```C#
public class CustomSignalExportExampleAlgorithm : QCAlgorithm
{
    public override void Initialize()
    {
        // In the Initialize method, set the custom signal export provider
        AddEquity("SPY");
        SignalExport.AddSignalExportProviders(new CustomSignalExport());
    }

    public override void OnData(Slice slice)
    {
        // Buy the SPY ETF when the algorithm is not invested
        if (!Portfolio.Invested)
        {
            SetHoldings("SPY", 0.1m);
        }
    }
}

// Define the custom signal export provider
public class CustomSignalExport : ISignalExportTarget
{
    private readonly HttpClient _httpClient = new ();

    public bool Send(SignalExportTargetParameters parameters)
    {
        // Create the request payload using the PortfolioTarget objects in the parameters
        // You can use the PortfolioTarget.Percent to calculate the target quantity
        // from the target percentage
        // The Tag property can be used to send additional information about the target
        // It's only available when you create the PortfolioTarget object with the tag parameter
        // See the Manual Exports documentation for more information
        object SimplePayload(PortfolioTarget target)
        {
            var newTarget = PortfolioTarget.Percent(parameters.Algorithm, target.Symbol, target.Quantity, tag:
target.Tag);
            return new { symbol = newTarget.Symbol.Value, quantity = newTarget.Quantity, tag = newTarget.Tag };
        };

        var message = JsonConvert.SerializeObject(parameters.Targets.Select(SimplePayload));
        using var httpMessage = new StringContent(message, Encoding.UTF8, "application/json");
        // Send the request to your custom endpoint
        var requestUri = "https://your-custom-endpoint.com/api/signal-export";
        using HttpResponseMessage response = _httpClient.PostAsync(requestUri, httpMessage).Result;
        var result = response.Content.ReadFromJsonAsync<RestResponse>().Result;

        // Log the result of the request for debugging purposes
        parameters.Algorithm.Log($"Send #{parameters.Targets.Count} targets. Success: {result.Success}");
        return result.Success;
    }

    public void Dispose() => _httpClient.Dispose();
}
```

Manual Exports

In some cases, you may need to send signals to your fund or a third-party manually. For example, the signal export manager sends signals after a fill event and you need to submit signals when the market is closed or your signals aren't based on your portfolio holdings.

To do this, pass a `PortfolioTarget` object or an array of `PortfolioTarget` objects to the `SetTargetPortfolio` method. The method returns a boolean that represents if the targets were successfully sent to its destination. In this situation, the number you pass to the `PortfolioTarget` constructor represents the portfolio weight. Don't use the `PortfolioTarget.Percent` method.

```C#
var target = new PortfolioTarget(_symbol, weight, tag: "My tag");
var success = SignalExport.SetTargetPortfolio(target);
```

If you use a margin account, you can send your current portfolio holdings by calling the `SetTargetPortfolioFromPortfolio`

method.

```C#
var success = SignalExport.SetTargetPortfolioFromPortfolio();
```

Signal Exports

Collective2

Introduction

Collective2 is a service that connects strategy developers and capital allocators. With our Collective2 integration, you can send trading signals from your live algorithms to Collective2 .

Add Providers

To export signals to Collective2 from your algorithm, during initialization , add a Collective2 signal export provider.

```
SignalExport.AddSignalExportProvider(new Collective2SignalExport(apiKey, systemId, useWhiteLabelApi));
```

The `Collective2SignalExport` constructor accepts the following arguments:

Argument: `apiKey` Your Collective2 API key. To get your API key, see Where can I find my API key? on the Collective2 website. Data Type: `string`
Argument: `systemId` Your Collective2 strategy Id. To get your strategy Id, see How do I find my strategy ID? on the Collective2 website. Data Type: `int`
Argument: `useWhiteLabelApi` Whether to use the white-label API instead of the general one. Data Type: `bool` | Default Value: `false`

You can add multiple signal export providers to a single algorithm.

Asset Classes

Our Collective2 integration supports signals for the following asset classes:

- US Equity
- Equity Options
- Forex
- Futures
- Index Options

Send Portfolio Targets

The signal export manager automatically sends signals when your portfolio holdings change to Collective2. By default, it waits five seconds after an order is filled to aggregate all the state changes into a single post. Set the `AutomaticExportTimeSpan` property to change the waiting time.

```
C#
SignalExport.AutomaticExportTimeSpan = TimeSpan.FromSeconds(1);
```

To send targets that aren't based on your portfolio holdings, see Manual Exports .

To disable automatic exports, set the `AutomaticExportTimeSpan` property to `null` .

```
C#
SignalExport.AutomaticExportTimeSpan = null;
```

Examples

Demonstration Algorithms
Collective2SignalExportDemonstrationAlgorithm.cs C# Collective2PortfolioSignalExportDemonstrationAlgorithm.cs C#

Signal Exports

Numerai

Introduction

Numerai is a platform where you can earn rewards by uploading unique live trading signals for a universe of US Equities. The regular Numerai Tournament is built around a free dataset they provide , but with Numerai Signals , you can now use any dataset to generate your trading signals. Through our Numerai integration, you can use our rich library of alternative datasets or your own custom datasets to create and automate your Numerai Signal submissions.

Add Providers

To export signals to Numerai from your algorithm, during initialization , add a Numerai signal export provider.

```C#
SignalExport.AddSignalExportProvider(new NumeraiSignalExport(publicId, secretId, modelId, fileName));
```

The `NumeraiSignalExport` constructor accepts the following arguments:

Argument: `publicId`

Your Numerai API key. To create and view your API keys, see the Automation section of your Account page on the Numerai website.

Data Type: `string` | Default Value: `None`

Argument: `secretId`

Your Numerai API secret. You get your API secret when you create a new API key.

Data Type: `string` | Default Value: `None`

Argument: `modelId`

The Id of the Numerai model. Follow these steps to get the model Id:

1. Open the Models page on the Numerai website.
2. In the Models section, click the three dots next to a model.
3. Click Copy Model ID .

Data Type: `string` | Default Value: `""`

Argument: `fileName`

The name for the signal file. If you use a file name that already exists, the new file overwrites the old one.

Data Type: `string` | Default Value: `"predictions.csv"`

You can add multiple signal export providers to a single algorithm.

Asset Classes

Our Numerai integration supports signals for US Equities .

Universe Selection

The Numerai Signals stock market universe covers roughly the top 5,000 largest stocks in the world. The universe available on QuantConnect that's the closest match to the Numerai Signals universe is the CRSP US Total Market Index, which represents approximately 100% of the investable US Equity market regularly traded on the New York Stock Exchange and Nasdaq. This Index doesn't contain all of the stocks in the Numerai Signals universe, but you don't need to submit signals for all the stocks in the Numerai Signals universe.

To get the constituents of the CRSP US Total Market Index, add an ETF constituents universe for the Vanguard Total Stock Market ETF, VTI.

```
UniverseSettings.Asynchronous = true;
_etfSymbol = AddEquity("VTI").Symbol;
AddUniverse(Universe.ETF(_etfSymbol));
```

Schedule Submissions

Every Tuesday, Wednesday, Thursday, Friday, and Saturday of the week, a new round is open for you to submit signals. Saturday rounds open at 18:00 UTC, and the submission window is open until Monday 14:30 UTC. Weekday rounds open at 13:00 UTC, and the submission window is open for 1 hour. For more information about the competition rounds, see Rounds in the Numerai documentation.

To schedule your submissions, you can set a Scheduled Event during initialization.

```
Schedule.On(
    DateRules.EveryDay(_etfSymbol),
    TimeRules.At(13, 0, TimeZones.Utc),
    SubmitSignals);
```

Send Portfolio Targets

To send targets, pass a list of `PortfolioTarget` objects to the `SetTargetPortfolio` method. The method returns a boolean that represents if the targets were successfully sent to Numerai Signals. In this situation, the number you pass to the `PortfolioTarget` constructor represents the portfolio weight. Don't use the `PortfolioTarget.Percent` method.

```
var success = SignalExport.SetTargetPortfolio(targets);
```

Signal Requirements

When you submit signals to Numerai, abide by the following rules:

- All signals must have a quantity between 0 and 1 (exclusive)
- You must submit at least 10 different signals per submission
- At least one of the weights in every batch of signals must be unique

Examples

Demonstration Algorithms
NumeraiSignalExportDemonstrationAlgorithm.cs C#

Signal Exports

vBase

Introduction

validityBase enables you to create a globally credible, independently verifiable audit trail for trading track records, models and signals. This ensures anyone viewing your results can confirm that your results are point-in-time, complete, free from cherry-picking and fully out-of-sample.

QuantConnect users can use validityBase to publish live, verifiable, indices of their trading strategies and signals - increasing their credibility with allocators, clients and potential employers, and significantly increasing their commercial value. Integration is simple. With a single function call from your QuantConnect code, you can automatically build an immutable blockchain-backed audit trail for your signals and models.

Add Providers

To export signals to vBase from your algorithm, during initialization , add a vBase signal export provider.

```
C#
SignalExport.AddSignalExportProvider(new VBaseSignalExport(apiKey, collectionName, storeStampedFile = true,
idempotent = false));
```

The `VBaseSignalExport` constructor accepts the following arguments:

Argument: `apiKey` Your vBase API key. You can find it in your profile Account Settings tab. Data Type: `string`
Argument: `collectionName` The name of the vBase collection to which the signals will be sent. You can create and manage collections in the vBase app Collections tab. Think of a collection like a strategy or dataset name. Data Type: `string`
Argument: `storeStampedFile` Whether to store a stamped copy of the signal in vBase. **Important** if this is marked "no", then you will be responsible for maintaining an exact copy of your stamped data. Losing or accidentally modifying stamped outputs will prevent your work from being verifiable. Data Type: `bool` | Default Value: `true`
Argument: `idempotent` Whether to make the signal export idempotent. If true, sending the same signal multiple times will only create one record in vBase. Data Type: `bool` | Default Value: `false`

You can add multiple signal export providers to a single algorithm.

Asset Classes

Our vBase integration supports all asset classes.

Send Portfolio Targets

To send targets, pass a list of `PortfolioTarget` objects to the `SetTargetPortfolio` method. The method returns a boolean value indicating whether the targets were successfully sent to vBase.

What gets stamped? The provider builds a `sym,wt` (symbol-weight) CSV and, under the hood, uses `PortfolioTarget.Percent` to convert your absolute target quantities into portfolio weights. This ensures the stamp reflects the weighted sizing of your positions at the time of export. Default behavior can be customized by overriding the `BuildCsv` method of the `VBaseSignalExport` class.

```C#
var success = SignalExport.SetTargetPortfolio(targets);
```

Examples

Demonstration Algorithms
VBaseSignalExportDemonstrationAlgorithm.cs C#

Strategy Library

Introduction

The Strategy Library is a collection of tutorials written by the QuantConnect team and community members. Review these tutorials to learn about trading strategies found in the academic literature and how to implement them with QuantConnect/LEAN.

Tutorials

Search Strategy Tutorials

Strategy Name
CAPM Alpha Ranking Strategy on Dow 30 Companies Applies CAPM model to rank Dow Jones 30 companies.
Combining Mean Reversion and Momentum in Forex Market Combines momentum and mean reversion techniques in the forex markets.
Pairs Trading-Copula vs Cointegration Applies Copula and Cointergration method to pairs trading.
The Dynamic Breakout II Strategy A demonstration of dynamic breakout II strategy.
Dual Thrust Trading Algorithm A demontration of Dual Thrust Intraday strategy.
Can Crude Oil Predict Equity Returns Applies regression method to predict the return from the stock market and compare it to the short-term U.S. T-bill rate.
Intraday Dynamic Pairs Trading using Correlation and Cointegration Approach A high frequency pairs trading algorithm based on cointegration.
The Momentum Strategy Based on the Low Frequency Component of Forex Market Applies high frequency filter to the momentum strategy.

Stock Selection Strategy Based on Fundamental Factors

MorningStar Fundamental factors universe selection algorithm.

Short-Term Reversal Strategy in Stocks

A short term reversal algorithm which gives the opposite signal by analyzing recent period price action.

Fundamental Factor Long Short Strategy

A basic monthly rebalance long short algorithm based on fundamental factors.

Asset Class Trend Following

Selects ETFs over ten-month moving average and assigns an equally weighted allocation.

Source: Quantpedia

Asset Class Momentum

Selects ETFs in different asset classes with the highest momentum and assigns an equally weighted allocation.

Source: Quantpedia

Residual Momentum

Constructs a long/short portfolio based on trailing residual momentum normalized by its standard deviation

Source: Quantpedia

Sector Momentum

Selects ETFs in different sectors with the highest momentum and assigns an equally weighted allocation.

Source: Quantpedia

Overnight Anomaly

Buy SPY ETF at its closing price and sell it at the opening each day.

Source: Quantpedia

Forex Carry Trade

Goes long the currency with the highest central bank interest rate and goes short the currency with the lowest interest rate.

Source: Quantpedia

Volatility Effect in Stocks

Constructs equally weighted portfolios by selecting stocks with the lowest volatility in the past one year.

Source: Quantpedia

Forex Momentum

Goes long currencies with strongest 12 month momentum against USD and goes short currencies with the lowest 12 month momentum against USD.

Source: Quantpedia

Pairs Trading with Stocks

Looks for the security that minimizes the sum of squared deviations and long-short position is opened when pair prices have diverged by multiple of standard deviations.

Source: Quantpedia

Short Term Reversal

Goes long stocks with the lowest return in the previous month and goes short stocks with the greatest return from the previous month.

Source: Quantpedia

Momentum Effect in Stocks

Goes long stocks with the best 12-month momentum in the large-cap universe.

Source: Quantpedia

Momentum Effect in Country Equity Indexes

Goes long stocks with the best 12-month momentum in the country equity indexes ETFs.

Source: Quantpedia

Mean Reversion Effect in Country Equity Indexes

Goes long country equity indexes ETFs with the worst 36-month return and short ETFs with the best 36-month return.

Source: Quantpedia

Liquidity Effect in Stocks

Goes long stocks with the lowest turnover and short on stocks with the highest turnover from the lowest market-cap quartile.

Source: Quantpedia

Volatility Risk Premium Effect

Sells at-the-money straddle with one month until maturity and buys an offsetting 15% out-of-the-money puts each month.

Source: Quantpedia

Momentum Effect in Commodities Futures

Goes long commodity futures with the highest momentum and short on futures with the lowest momentum.

Source: Quantpedia

Small Capitalization Stocks Premium Anomaly

Goes long stocks with the lowest market capitalization and rebalances the portfolio once a year.

Source: Quantpedia

Paired Switching

Goes long asset with better performance over the last period and rebalances portfolio every quarter.

Source: Quantpedia

Term Structure Effect in Commodities

Buys each month the 20% of commodities with the highest roll-returns and shorts the 20% of commodities with the lowest roll-returns and holds the long-short positions for one month.

Source: Quantpedia

Momentum Effect Combined with Term Structure in Commodities

Portfolios are formed based on roll returns and the algorithm goes long and short contracts with the highest and lowest one-month performance.

Source: Quantpedia

Book-to-Market Value Anomaly

Quintile portfolios are formed based on the Book-to-Market ratio and the highest quintile is held for one year.

Source: Quantpedia

Gold Market Timing

Goes long gold when the Fed model shows that the market is undervalued (the earnings yield is higher than the bond yield and their ratio is at least 2).

Source: Quantpedia

Turn of the Month in Equity Indexes

Buys SPY the day before the end of the month and liquidates position on 3rd trading day of new month.

Source: Quantpedia

Momentum - Short Term Reversal Strategy

Goes long stocks with the decreasing return from the winner group and short stocks with the increasing return from the loser group.

Source: Quantpedia

Pairs Trading with Country ETFs

Identifies the price divergence from two highly correlated country ETFs and takes a market neutral position.

Source: Quantpedia

Sentiment and Style Rotation Effect in Stocks

Creates long-short positions of growth and value stocks based on the investment sentiment.

Source: Quantpedia

Asset Growth Effect

Creates long-short positions of stocks based on the annual change of their total assets.

Source: Quantpedia

Momentum and State of Market Filters

Goes long and short stocks with the highest and lowest six-month momentum respectively if the previous 12 months return on the broad market index was positive.

Source: Quantpedia

Accrual Anomaly

Decile portfolios are formed based on balance sheet based accruals and highest decile is shorted while lowest decile is bought for a year.

Source: Quantpedia

Momentum in Mutual Fund Returns

Forms a long-short portfolio of asset management firms based on trailing rate of change and nearness to trailing high.

Source: Quantpedia

Momentum and Style Rotation Effect

Goes long style index ETF with the highest 12-month momentum and short ETF with the lowest 12-month momentum.

Source: Quantpedia

Trading with WTI BRENT Spread

Goes long the spread if the spread is below 20-day moving average and short if the spread is above 20-day moving average.

Source: Quantpedia

Momentum Effect in REITs

Tercile portfolios are formed based on momentum and the best performing portfolio is held.

Source: Quantpedia

Option Expiration Week Effect

Goes long S&P 100 index ETF during option expiration week and stays in cash during other days.

Source: Quantpedia

Earnings Quality Factor

Goes long stocks with high earnings quality and short stocks with low earnings quality based on composite factor score.

Source: Quantpedia

January Effect in Stocks

Invests into small cap stocks at the beginning of each January and stays invested in large cap stocks for rest of the year.

Source: Quantpedia

Momentum and Reversal Combined with Volatility Effect in Stocks

Goes long on stocks from the highest performing quintile from the highest volatility group and short on stocks from the lowest performing quintile from the highest volatility group.

Source: Quantpedia

ROA Effect within Stocks

Goes long on stocks with highest ROA and short stocks with the lowest ROA from each market capitalization group.

Source: Quantpedia

January Barometer

Invested in equity market with ETF only if January return is positive otherwise switch investments to T-Bills.

Source: Quantpedia

Lunar Cycle in Equity Market

Goes long in emerging market index ETF 7 days before the new moon and switch to a short position on emerging market index ETF 7 days before the full moon.

Source: Quantpedia

VIX Predicts Stock Index Returns

Goes long on equity index ETF if the VIX is in the highest percentile short if VIX is in the lowest percentile in the last two-year history.

Source: Quantpedia

Combining Momentum Effect with Volume

Goes long stocks with the highest volume from the top momentum decile and short stocks with the highest volume from the bottom momentum decile.

Source: Quantpedia

Short Term Reversal with Futures

Goes long (short) on futures from the high-volume, low-open interest group with the lowest (greatest) returns in the previous week.

Source: Quantpedia

Pre-holiday Effect

Invests in equity market 2 days preceding holiday days and stays in cash during the other trading days.

Source: Quantpedia

Beta Factors in Stocks

Goes long stocks with the bottom beta and short stocks with the top beta, securities are weighted by the ranked betas.

Source: Quantpedia

Exploiting Term Structure of VIX Futures

Buys or sells the nearest VIX futures based on the daily roll and hedge against the open positions with E-mini S&P500 futures.

Source: Quantpedia

12 Month Cycle in Cross-Section of Stocks Returns

Reviews the returns from last January, going long on the top 10% winners and short the bottom 10%.

Source: Quantpedia

Momentum Effect in Stocks in Small Portfolios

Goes long in the 10 stocks with the highest performance and goes short in the 10 stocks with the lowest performance in the previous one year.

Source: Quantpedia

Value Effect within Countries

Invests in the cheapest 33% of country ETFs according to CAPE ratios.

Source: Quantpedia

Beta Factor in Country Equity Indexes

Goes long on the low-beta portfolio and short on the high-beta portfolio in country indexes ETFs.

Source: Quantpedia

Price to Earnings Anomaly

Invests in stocks with low P/E ratio.

Fama French Five Factors

Stock selecting strategy based on Fama-French Five Factors Model.

Source: NYU

Mean-Reversion Statistical Arbitrage Strategy in Stocks

Apply statistical arbitrage to take advantage of pricing inefficiencies in stocks.

Source: NYU

Expected Idiosyncratic Skewness

Stock selection strategy that calculates expected idiosyncratic skewness using Fama-French three-factor model, sorts stocks based on the calculated skewness, and longs the bottom 5%.

Source: NYU

Risk Premia in Forex Markets

A strategy based on asymmetric tail risks and excess returns in forex markets.

Source: NYU

Seasonality Effect based on Same-Calendar Month Returns

A strategy that takes long and short positions based on historical same-calendar month returns

Source: NYU

Standardized Unexpected Earnings

Stock selection strategy that calculates the unexpected earnings, standardizes the unexpected earnings, goes long on the top 5%, and rebalances the portfolio monthly.

Source: NYU

Price and Earnings Momentum

A momentum strategy based on quarterly returns and earnings growth

Source: NYU

Improved Momentum Strategy on Commodities Futures

An advanced momentum strategy that modifies the basic momentum strategies by introducing Baltas and Kosowski weights and rebalances the portfolio monthly. The new weighing scheme incorporates trend strength into the trading signal, uses an efficient volatility estimator, and adds a dynamic leverage mechanism.

Source: NYU

Commodities Futures Trend Following

A simple trend following strategy on commodities futures.

Source: NYU

Forecasting Stock Prices using a Temporal CNN Model

Applying a Temporal Convolutional Neural Network to forecasting future stock prices.

Source: Tampere University

Leveraged ETFs with Systematic Risk Management

We apply Simple Moving Averages to manage the risk of holding leveraged ETFs in an attempt to beat the S&P500

Source: The Lead-Lag Report

Ichimoku Clouds in the Energy Sector

A techincal indicator crossover strategy trading the largest energy companies.

Source: SSRN

Intraday ETF Momentum

A momentum strategy based on returns of the market open

Source: NYU

Intraday Arbitrage Between Index ETFs

A strategy that tracks the price paths of two correlated ETFs and takes advantage of mis-pricings that arise when the price paths diverge

Source: SSRN

Optimal Pairs Trading

Mathematically Deriving the Optimal Entry and Liquidation Values of a Pairs Trading Process

Source: arXiv

G-Score Investing

Applying G-Score Investing to Invest in a Portfolio of Technology Stocks

Source: SSRN

SVM Wavelet Forecasting

Forecasting EURJPY prices with an SVM Wavelet model

Source: Academia

Gradient Boosting Model

Forecasts future intraday returns with a gradient boosting model trained on technical indicators

Source: arXiv

Using News Sentiment to Predict Price Direction of Drug Manufacturers

Analyzes the news releases of drug manufacturers and places intraday trades for the stocks with positive news.

Source: arXiv

Gaussian Naive Bayes Model

Forecasts the next day's return of technology stocks by fitting a gaussian naive bayes model to the historical returns of the technology sector constituents.

Source: Academia

API Reference

The QuantConnect API Reference is a comprehensive technical document that details every class, method, property, and event available in the LEAN algorithmic trading framework. It serves as the definitive source for understanding how to interact with LEAN, helping you quickly look up syntax and explore available features. The LEAN engine is written in C#, but you can create algorithms with either C# or Python since LEAN uses Python.Net to bridge between your Python code and the underlying C# engine. For your convenience, we provide an API Reference for both languages.

Python

C#

Migrations

Migrations

Zipline

The following pages explain how to migrate trading algorithms from Zipline to LEAN:

Initialization

Using Data

Ordering

Logging and Plotting

Quick Reference

Zipline

Initialization

Initializing State

On QuantConnect, algorithm initialization behavior is equivalent to Quantopian's `initialize` method. `Initialize` is the method equivalent, and it is only ran once at the start of the algorithm. It is responsible for initially adding data to the algorithm, adding a universe, setting the brokerage, setting the algorithm's start and end date, etc.

The algorithm structure in QuantConnect differs from Quantopian's as well. On QuantConnect, we create a class that derives from `QCAlgorithm` and define our overrides/algorithm functionality there. An example of an algorithm with initialization is provided.

Quantopian

QuantConnect

On QuantConnect, the `self` parameter is the equivalent of the `context` parameter in Quantopian algorithms. You can use `self` to access algorithm resources, such as the Portfolio, Orders, Order Tickets, Securities, place orders in your algorithm, and much more.

You can read more about algorithm initialization at the Initializion documentation page.

Attaching Pipelines

The functional equivalent of Pipelines on QuantConnect are Universes. Universes can be used to dynamically select assets your algorithm wants to trade or request data from. To add a universe to your algorithm, you can call the `AddUniverse()` method and provide the required Coarse and Fine universe selection functions.

Coarse Universe selection is the first step of universe selection. It provides a means to filter data in a lightweight fashion before proceeding to Fine Universe selection. Coarse universe will allow you to filter assets based off top-level fundamental factors, such as Dollar Volume for the day, if a company has fundamental data, and market of the asset.

Fine Universe selection is the second step of universe selection. It provides fundamental data to your user-defined function, and can be used to filter in greater detail based on an asset's fundamental data, such as earnings, EPS, etc.

An example of coarse and fine universe selection to filter assets in our algorithm is provided. Adding universes can be used as an alternative to using `AddEquity` if you prefer dynamic asset selection.

You can configure universe settings by modifying the existing `UniverseSettings` attribute of the algorithm.

You can read more about universe selection at the Universe documentation page.

Scheduling Functions

Scheduled Events on QuantConnect are the equivalent of `schedule_function()` from Quantopian. Examples are provided comparing the Scheduled Events functionality between QuantConnect and Quantopian.

Quantopian

QuantConnect

You can read more about scheduled events at the Scheduled Events documentation page.

Setting Slippage and Commissions

QuantConnect supports applying slippage and commissions per individual security.

To add a slippage model to a security, we must set the `SlippageModel` property of the `Security` object to our model. A `Security` is returned when data is added to the algorithm. Examples of setting slippage in an algorithm is provided below for both Quantopian and QuantConnect.

Quantopian

QuantConnect

Commissions

To add a fee model (otherwise known as Commissions on Quantopian) to a security, we must set the `FeeModel` property of the `Security` object to our model. Examples are provided below for Quantopian and QuantConnect.

Quantopian

QuantConnect

You can read more about Securities and their attributes at the Securities and Portfolio and Reality Modeling documentation pages.

Manual Asset Lookup

On QuantConnect, we provide the `Symbol` class to reference a security across time with a changing ticker. This is similar to Quantopian's `symbol()` function, which enables the same functionality.

When you add data to your algorithm, a `Security` object is returned. The `Symbol` is accessible from the `Security` as shown below.

To manually create a reference to an old equity, you can use the `Symbol.create()` method.

You can read more about Symbols and Security Identifiers (SID) at the Security Identifiers documentation page.

Zipline

Using Data

Pipeline in Algorithms

As covered in the Initialization section, Pipelines can be replicated using Universe Selection in QuantConnect, albeit with some additional steps in between and a performance impact.

In this section, we will construct and define an equivalent pipeline model using universe selection in QuantConnect. We will filter our data set in Coarse and Fine, and apply an additional filter to historical data with a rolling window.

We first create a skeleton algorithm definition to begin setting up our Pipeline. Note that this algorithm will allow all equities through, which will have a substantial performance impact on our algorithm.

The skeleton algorithm is the equivalent of the Pipeline call below.

The equivalent of `Pipeline(screen=...)` resolves to the filter applied at the Coarse and Fine stages of universe selection. Let's define a filter of stocks with a dollar volume greater than $50000000 USD, as well as a rolling thirty day return greater than 2%. Once we've initially filtered the Symbols in Coarse Universe Selection, let's define a final filter only allowing stocks with EPS greater than 0. Beware of making `History()` calls with many Symbols. It could potentially cause your algorithm to run out of system resources (i.e. RAM) and reduce performance of your algorithm on universe selection.

This class definition is now roughly equivalent to the following `CustomFactor` and Pipeline call in Quantopian, excluding the EPS filtering.

An example of the shape of the DataFrame returned from History is shown below. The DataFrame has a MultiIndex, with `level=0` being the Symbol, and `level=1` being the Time for that point of data. You can index the Symbol/ `level=0` index by using either the SecurityIdentifier string (e.g. `df.loc["AAPL R735QTJ8XC9X"]`) or with the ticker of the Symbol (e.g. `df.loc["AAPL"]`)

		close	high	low	open	volume
symbol	time					
IBM	2019-04-26	138.62	139.82	137.70	139.82	1419038.0
	2019-04-27	139.44	139.89	138.81	139.28	1275702.0
AAPL	2019-04-26	205.24	207.76	205.12	206.84	10105760.0
	2019-04-27	204.29	205.00	202.12	204.83	9984854.0

BarData Lookup

Similar but different, the Quantopian `BarData` object, and the QuantConnect `Slice` object both provide data to the user's algorithm as point-in-time data.

In Quantopian, data is handled via the `handle_data(context, data)` function. In QuantConnect, data is handled via the `OnData(self, slice)` method. Both of these functions accept data whenever new data exists for a given point in time. Although these two functions share the same method signature, the handling of the data is different.

`BarData` vs. `Slice`

`BarData` is the primary mechanism to retrieve the point-in-time data, as well as requesting history for any given securities in Quantopian. The following code retrieves daily historical data from 30 days into the past, as well as getting the most recent data

for AAPL at the current point-in-time.

Quantopian

QuantConnect

`Slice` represents a single point in time, and does not provide the functionality to access historical data itself. To access historical data in an algorithm, use the algorithm's `self.History()` call to request a pandas DataFrame of historical data. In `Slice`, the data that is accessed is not a pandas DataFrame, but rather a single object containing data for a single point in time.

The `TradeBar` class, for example, only contains scalar values of OHLCV, rather than return a DataFrame of OHLCV values. Since the data `Slice` contains is point-in-time, there will be only a single trade/quote bar per Symbol whenever `OnData(self, data)` is called.

QuantConnect provides Quote (NBBO) data for use in your algorithm, otherwise known as a `QuoteBar`. Quote data is only accessible when an algorithm is set to consume Tick, Second, or Minutely data.

You can access Trade (OHLCV) data by accessing the `Bars` attribute of `Slice`. You can access Quote (Bid(OHLCV), Ask(OHLCV)) data by accessing the `QuoteBars` attribute of `Slice`.

Both of the `Bars` and `QuoteBars` attributes are similar to Python dictionaries, and can be used as such. To check to see if there exists a new piece of data for a given security, you can use Python's `in` operator on `Bars` and or `QuoteBars`. You can also choose to iterate on all data received by calling the `Values` attribute of the `Bars` or `QuoteBars` attributes, which will return either a list of `TradeBar` or `QuoteBar` objects.

The `TradeBar` object contains the `Open`, `High`, `Low`, `Close`, `Volume`, `Time`, `EndTime`, and `Symbol` attributes. The `QuoteBar` object contains the following attributes:

- `Bid.Open`, `Bid.High`, `Bid.Low`, `Bid.Close`, `LastBidSize`
- `Ask.Open`, `Ask.High`, `Ask.Low`, `Ask.Close`, `LastAskSize`
- `Time`, `EndTime`, and `Symbol`.

Note that the `Bid` and `Ask` attributes can potentially be `None` if no bid/ask data exists at a given point-in-time.

The example below shows the different ways to access `TradeBar` and `QuoteBar` data, as well as requesting 30 days of AAPL historical data.

Zipline

Ordering

Placing Orders

Both Quantopian and QuantConnect offer several methods for placing orders. Both platforms have `MarketOrder` , `LimitOrder` , `StopOrder` , and `StopLimitOrder` classes to create specific order types, although the `StopOrder` order is named `StopMarketOrder` on our platform. On Quantopian, creating these orders is done by passing a `style` argument to the `order` method.

On QuantConnect, the same orders can be created with

In addition to the order types above, QuantConnect has several other order types available. Refer to our Trading and Orders documentation for a comprehensive list.

Quantopian's `order_optimal_portfolio` method computes the optimal portfolio weights using an objective and constraints, then places the orders to achieve the desired portfolio. For example, the code below uses the Quantopian API to create an equal-weighted dollar-neutral portfolio.

To acheive the same functionality with QuantConnect, we utilize portfolio construction models . For instance, to create an equal-weighted dollar-neutral portfolio, we could define the following portfolio construction model and attach it to the algorithm.

When algorithms require manual control of their orders, both Quantopian and QuantConnect have lower-level ordering methods. Quantopian's `order` method, which places an order for a fixed number of shares, directly maps to our `Order` method. Additionally, Quantopian's `order_target_percent` method places an order to adjust a position to a target percent of the current portfolio value. Here's an example use case of these methods on the Quantopian platform.

On QuantConnect, these same orders can be placed using the `SetHoldings` and `Order` methods.

Quantopian and QuantConnect don't have equivalents for all the ordering techniques, although we can create a workaround for most situations. For instance, Quantopian's `order_percent` method places an order in the specified asset corresponding to the given percent of the current portfolio value.

To accomplish this on QuantConnect, we can determine the weight of the asset in the `Portfolio` , then manually determine the new target percent to pass to `SetHoldings` .

The Porfolio Object

Quantopian's `Portfolio` class provides read-only access to the current portfolio state. The state includes starting cash, current cash, portfolio value, and positions. The `Portfolio` object is accessed through the `context` object, as seen below.

All of this information, and more, is attainable on QuantConnect by using our `Portfolio` object. Listed below is an example of doing so. Although there isn't a property for starting cash, this value can be saved during Initialization to be referenced later.

On Quantopian, by iterating through the `positions` dictionary, we can access information about currently-held positions. This includes the number of shares held, the last traded price & date of the asset, and the position's cost basis.

Since the `Portfolio` on QuantConnect is a `Dictionary<Symbol, SecurityHolding>` , we can get information on current positions by indexing the `Portfolio` object with a `Symbol` or ticker. The `SecurityHolding` object that is returned contains

information related to a single security in the portfolio. For instance, we have

For more information on our `Portfolio` object refer to the Securities and Portfolio documentation.

Zipline

Logging and Plotting

Logging

In QuantConnect, the `self.Debug()` method is the equivalent of the `log.info()` function in Quantopian. Debug output is viewable via the QuantConnect Terminal Console.

Output to the console via `self.Debug()` is rate-limited. If you need long-term storage of your logs and don't want to overflow the console, consider using `self.Log()` instead. An example comparing logging between Quantopian and QuantConnect is shown below.

Quantopian

QuantConnect

Logging via `self.Log()` has daily limits in QuantConnect. On the free tier, users are limited to 10KB of logs per day for your account/organization. To increase this limit, you can upgrade your account to a different tier. Calls to `self.Debug()` will also count towards the 10KB log limit, but are always outputted to the Terminal Console regardless of your daily log limit.

To view pricing and upgrade your account, visit the Organization Pricing page.

You can view the logs generated via `self.Log()` for a backtest by first accessing a backtest, and clicking the "Logs" tab near the bottom of the page. To download the logs, you can click the "Download Logs" link to download a text file containing your backtest's logs.

Plotting

In Quantopian, plotting/charting is done with the `record(series1_name=scalar_value, series2_name=scalar_value, ...)` function. In QuantConnect, plotting is accessible via the `self.plot("chart name", "series name", scalar_value)` method.

Plotting multiple series in one function call is possible in Quantopian. However, in QuantConnect, a separate call is needed for each series requiring an update. All series with the same chart name will appear on the same window/pane. New charts can be created by calling `self.plot()` method with a unique chart name.

An example of plotting in Quantopian vs. QuantConnect is shown below. The QuantConnect code will create two charts, one named "Chart1" with two series of data, and another named "Chart2" with one series of data. Note that you can also use `self.plot("series name", value)` to place the custom series on the same chart as the equity curve.

Quantopian

QuantConnect

For more information on plotting in QuantConnect, visit the Charting documentation page.

Zipline

Quick Reference

User-Implemented Functions

The following table describes functions that are treated specially when defined in the IDE.

Algorithms are required to implement one method: Initialize(self).

An optional scheduled event can be defined to behave like before_trading_start(). OnData() is also optional but is treated especially if defined.

Quantopian	**QuantConnect**
`initialize` (context)	`Initialize` (self)
Required function called once at the start of a backtest.	
`before_trading_start` (context, data)	`self.Schedule.On(self.DateRules.EveryDay(), self.TimeRules.AfterMarketOpen(symbol, 10), self.before_trading_start)`
Optional function called prior to market open on every trading day.	
`handle_data` (context, data)	`OnData` (self, slice)
Optional function called at the end of each market minute.	

Interface Classes

The following table describes classes that are passed to `initialize()` , `handle_data()` , `before_trading_start()` and any scheduled with `schedule_function()` .

In QuantConnect/Lean, the portfolio state and the positions are class attributes of QCAlgorithm, available via self keyword. The scheduled function is parameterless and can algorithms can self.CurrentSlice.

Quantopian	QuantConnect
`quantopian.algorithm.interface.AlgorithmContext`	`QCAlgorithm`
Shared object for storing custom state.	
`quantopian.algorithm.interface.BarData`	`Slice`
Provides methods for accessing minutely and daily price/volume data from Algorithm API functions.	
`quantopian.algorithm.interface.Portfolio`	`self.Portfolio`
Object providing read-only access to current portfolio state.	
`quantopian.algorithm.interface.Positions`	`self.Transactions`
A dict-like object containing the algorithm's current positions.	

The features from the BarData methods are found in the Slice object (current data), History method (past data) and class attribute Securities.

Quantopian	QuantConnect
`quantopian.algorithm.interface.BarData.current`	`Slice[symbol]`
Returns the "current" value of the given fields for the given assets at the current simulation time.	
`quantopian.algorithm.interface.BarData.history`	`self.History(symbols, bar_count)`
Returns a trailing window of length `bar_count` containing data for the given assets, fields, and frequency.	
`quantopian.algorithm.interface.BarData.can_trade`	`self.Securities[symbol].IsTradable`
For the given asset or iterable of assets, returns True if all of the following are true:	
`quantopian.algorithm.interface.BarData.is_stale`	`not Slice[symbol].IsFillForward`
For the given asset or iterable of assets, returns True if the asset is alive and there is no trade data for the current simulation time.	

Pipeline

The following table describes functions used to schedule and retrieve results of Universe selection. For more information on the Universe Selection API, see the Univeres API Reference.

Quantopian	QuantConnect
`quantopian.algorithm.attach_pipeline` (...)	`universe = self.AddUniverse` (...) name = universe.Configuration.Symbol
Register a pipeline to be computed at the start of each day.	
`quantopian.algorithm.pipeline_output` (name)	`self.UniverseManager` [name]
Get results of the pipeline attached by with `name` .	

Scheduling Functions

The following tables describes functions that can be used to schedule functions to run periodically during your algorithm.

Quantopian	QuantConnect
`quantopian.algorithm.schedule_function` (func)	`self.Schedule.On(IDateRule, ITimeRule, func)`
Schedule a function to be called repeatedly in the future.	

Date Rules

The following table contains functions that can be used with the `date_rule` parameter of `schedule_function()` .

Quantopian	QuantConnect
`quantopian.algorithm.date_rules.every_day` ()	`self.DateRules.EveryDay(symbol)`
Create a rule that triggers every day.	
`quantopian.algorithm.date_rules.month_start` ([...])	`self.DateRules.MonthStart(symbol, daysOffset)`
Create a rule that triggers a fixed number of trading days after the start of each month.	
`quantopian.algorithm.date_rules.month_end` ([...])	`self.DateRules.MonthEnd(symbol, daysOffset)`
Create a rule that triggers a fixed number of trading days before the end of each month.	
`quantopian.algorithm.date_rules.week_start` ([...])	`self.DateRules.WeekStart(symbol, daysOffset)`
Create a rule that triggers a fixed number of trading days after the start of each week.	
`quantopian.algorithm.date_rules.week_end` ([...])	`self.DateRules.WeekEnd(symbol, daysOffset)`
Create a rule that triggers a fixed number of trading days before the end of each week.	

Time Rules

The following table contains functions that can be used with the `time_rule` parameter of `schedule_function()` .

Quantopian	QuantConnect
`quantopian.algorithm.time_rules.market_open` ([...])	`self.TimeRules.AfterMarketOpen(symbol, minutes_after_open = 0)`
Create a rule that triggers at a fixed offset from market open.	
`quantopian.algorithm.time_rules.market_close` ([...])	`self.TimeRules.BeforeMarketClose(symbol, minutes_before_close = 0)`
Create a rule that triggers at a fixed offset from market close.	

Ordering

There are several ways to place orders from an algorithm. For most use-cases, we recommend using SetHoldings method which correctly accounts the porfolio buying power model and fees. At the moment, there is no equivalent for `order_optimal_portfolio()` . However a similar behavior can be replicated with a Portfolio Construction Model.

Algorithms that require explicit manual control of their orders can use the lower-level ordering functions.

Placing Orders

Quantopian	QuantConnect
`quantopian.algorithm.order_optimal_portfolio` (...)	`N/A`
Calculate an optimal portfolio and place orders toward that portfolio.	
`quantopian.algorithm.order` (asset, amount[, ...])	`self.Order(symbol, amount)`
Place an order for a fixed number of shares.	
`quantopian.algorithm.order_value` (asset, value)	`N/A`
Place an order for a fixed amount of money.	
`quantopian.algorithm.order_percent` (asset, ...)	`self.SetHoldings(symbol, percent)`
Place an order in the specified asset corresponding to the given percent of the current portfolio value.	
`quantopian.algorithm.order_target` (asset, target)	`N/A`
Place an order to adjust a position to a target number of shares.	
`quantopian.algorithm.order_target_value` (...)	`N/A`
Place an order to adjust a position to a target value.	
`quantopian.algorithm.order_target_percent` (...)	`self.SetHoldings(symbol, percent)`
Place an order to adjust a position to a target percent of the current portfolio value.	

Controlling Order Execution

Quantopian	QuantConnect
`zipline.finance.execution.ExecutionStyle`	`FillModel`
Base class for order execution styles.	
`zipline.finance.execution.MarketOrder` ([exchange])	`self.MarketOrder` (symbol, quantity)
Execution style for orders to be filled at current market price.	
`zipline.finance.execution.LimitOrder` (limit_price)	`self.LimitOrder` (symbol, quantity, limit_price)
Execution style for orders to be filled at a price equal to or better than a specified limit price.	
`zipline.finance.execution.StopOrder` (stop_price)	`self.StopMarketOrder` (symbol, quantity, stop_price)
Execution style representing a market order to be placed if market price reaches a threshold.	
`zipline.finance.execution.StopLimitOrder` (...)	`self.StopLimitOrder` (symbol, quantity, stop_price, limit_price)
Execution style representing a limit order to be placed if market price reaches a threshold.	

Managing Existing Orders

Quantopian	QuantConnect
`quantopian.algorithm.get_order` (order_id)	`self.Transactions.GetOrderById` (order_id)
Lookup an order based on the order Id returned from one of the order functions.	
`quantopian.algorithm.get_open_orders` ([asset])	`self.Transactions.GetOpenOrders` (symbol)
Retrieve all of the current open orders.	
`quantopian.algorithm.cancel_order` (order_param)	`self.Transactions.CancelOpenOrders` (symbol)
Cancel an open order.	

Customizing the Simulation

Slippage

Algorithms can customize the simulation of order fills by calling SetSlippageModel() with a ISlippageModel which is any class that implements GetSlippageApproximation method. Unlike Quantopian, the slippage model is a security attribute, not a global attribute. Different securities can have different slippage models.

The default slippage model on QuantConnect is a constant slippage model with zero slippage percent (no slippage).

Quantopian	QuantConnect
`quantopian.algorithm.set_slippage` ([...])	`self.Securities[symbol].SetSlippageModel(ISlippageModel)`
Set the slippage models for the simulation.	
`zipline.finance.slippage.SlippageModel` ()	`N/A (class must implement GetSlippageApproximation method)`
Abstract base class for slippage models.	
`zipline.finance.slippage.FixedBasisPointsSlippage` ([...])	`ConstantSlippageModel(slippage_percent)`
Model slippage as a fixed percentage difference from historical minutely close price, limiting the size of fills to a fixed percentage of historical minutely volume.	
`zipline.finance.slippage.NoSlippage` ()	`NullSlippageModel.Instance`
A slippage model where all orders fill immediately and completely at the current close price.	
`zipline.finance.slippage.FixedSlippage` ([spread])	`N/A`
Simple model assuming a fixed-size spread for all assets.	
`zipline.finance.slippage.VolumeShareSlippage` ([...])	`N/A`
Model slippage as a quadratic function of percentage of historical volume.	

Commissions

Algorithms can customize the simulation of order fees by calling SetFeeModel() with a IFeeModel which is any class that implements GetOrderFee method. Unlike Quantopian, the fee model is a security attribute, not a global attribute. Different securities can have different fee models.

The default fee model on QuantConnect depends on the security type and the brokerage. For instance, the default fee model for equity is the Interactive Brokers fee model.

Quantopian	QuantConnect
`quantopian.algorithm.set_commission` ([...])	`self.Securities[symbol].SetFeeModel(IFeeModel)`
Sets the commission models for the simulation.	
`zipline.finance.commission.CommissionModel`	`N/A (class must implement GetOrderFee method)`
Abstract base class for commission models.	
`zipline.finance.commission.PerShare` ([cost, ...])	`N/A`
Calculates a commission for a transaction based on a per share cost with an optional minimum cost per trade.	
`zipline.finance.commission.PerTrade` ([cost])	`N/A`
Calculates a commission for a transaction based on a per trade cost.	
`zipline.finance.commission.PerDollar` ([cost])	`N/A`
Model commissions by applying a fixed cost per dollar transacted.	
`zipline.finance.commission.NoCommission`	`FeeModel()`
Model commissions as free.	

Benchmark

The default benchmark is SPY.

Quantopian	QuantConnect
`quantopian.algorithm.set_benchmark` (benchmark)	`self.SetBenchmark` (benchmark)
Set the benchmark asset.	

Processing math: 25%

Processing math: 35%